SPECTRUM

Melbourne House

SPECTRUM MACHINE LANGUAGE FOR THE ABSOLUTE BEGINNER

Edited by William Tang

SPECTRUM MACHINE LANGUAGE FOR THE ABSOLUTE BEGINNER

Edited by
William Tang

Melbourne House Publishers

Published in the United Kingdom by:
Melbourne House (Publishers) Ltd.,
Church Yard,
Tring, Hertfordshire HP23 5LU
ISBN 0 86161 110 1

Published in Australia by:
Melbourne House (Australia) Pty. Ltd.,
Suite 4, 75 Palmerston Crescent,
South Melbourne, Victoria 3205.

Published in the United States of America by:
Melbourne House Software Inc.,
347 Reedwood Drive,
Nashville TN 37217.

Printed in Hong Kong by Colorcraft Ltd.

D C B A 9 8 7 6 5 4 3 2 1

Contents

Finding Your way around Machine Language:

The Beginning 5
Basic Machine Language Concepts 11
The Way Computers Count 18
How Information is Represented 24
A Look into the CPU 30
This is All Very Well ... 39

How the CPU Uses its Limbs 43
Counting Off Numbers on One Hand 51
Flags and their Uses 58
Counting Up and Down 64
One Handed Arithmetic 69
Logical Operators 75

Coping with Two Handed Numbers 79
Manipulating Numbers with Two Hands 83
Manipulating the Stack 91
Two fisted arithmetic 95
Loops and Jumps 99
Use of Subroutines 106
Block Operations 109

Instructions That are Less Frequently Used

Register Exchanges 115
Bit, Set and Reset 117
Rotates and Shifts 119
In and Out 122
BCD Representation 126
Interrupts 127
Restarts 128

Programming Your Spectrum

Planning Your Program 130
Features of the Spectrum 135

Monitor Programs

EZ-Code Machine Language Editor 145
HexLoad Machine Code Monitor 155

The FREEWAY FROG Program

Program Design 161
Stage 1 - Data base 164
Stage 2 - Initialisation 172
Stage 3 - Regular Traffic 176
Stage 4 - Police Car 181
Stage 5 - The Frog 185
Stage 6 - Control 190

Appendices:

Spectrum key Input Table 227
Screen display Map 228
Character set Table 229
Decimal/Hexadecimal conversions 230
Falg Operations Summary 234
Z80 Instructions by op-code 236
Z80 Instructions by mnemonics 240

The Beginning

This book is designed as an introduction to the field of machine and assembly language programming for the "Sinclair ZX Spectrum."

It may be that you are coming to this book with no clear idea of what machine language programming is all about.

You may not even know what machine language is. You may not even be aware that there is a difference between machine language and assembly language, nor indeed how they differ from programming in BASIC.

Don't worry, and don't be frightened by the jargon - we will explain everything step by step.

First, let us look at the way a computer operates:

```
Programmer ---------------> Keyboard
    ^                          |
    |                          v
TV Screen <--------------- Operating System
                                ^
                                 \
                                  v
                             Central
                                 Processing
                                    Unit
```

What this diagram aims to show is that there is a barrier between the programmer and the brain of the Spectrum, the Central Processing Unit. It is not possible under normal processing for the programmer to tell the Central Processing Unit - usually referred to as the CPU - what to do.

In the Sinclair machines, the CPU chosen is the Z80A chip, which is a faster version of the popular Z80 chip. There are 4 chips - Z80, 6502, 6809, and 8088 - which have become widely accepted as CPUs for microcomputers. The Z80 is by far the most popular chip.

I am sure it comes as no surprise to learn that the Z80A does not understand a word of 'BASIC'! Indeed no CPU has been designed so that we can communicate directly with the brain of the computer.

If you think about it for long enough, you will realise that it would be very difficult, if not impossible, in any case to give a chip in a computer an instruction that would make any sense to a human. Take the top off your Sinclair (if you dare!) and have a look at the chip nearest to the speaker - this is the Z80A CPU. Obviously this chip in your computer can only respond to electrical signals that are passed on to it by the rest of the circuitry!

What is Machine Language?

The Z80 chip has been designed in such a way that it can accept signals simultaneously from eight of the pins connected to it.

The designers of the Z80 chip constructed it in such a way that different combinations of signals to the Z80 chip along these eight pins would 'instruct' the Z80 to perform different functions.

Keeping in mind that what is really happening are electrical signals, let's adopt a convention to represent these signals - for example showing a '1' if there is a signal to one of the pins, or a '0'if there is no signal.

A typical instruction might therefore look something like:

0 0 1 1 1 1 0 0

Quite a long way from something like
'Let A = A + 1',
for example, isn't it!

Nonetheless, this is what machine language is all about. The name says it all! It is a language for machines. Each manufacturer of the different chips has designed a different 'language' for its products!

At this stage you may be asking yourself - if this is what machine language is all about, why bother? Why not accept the benefits of someone else's work which allows me to program the computer in a language I can easily understand, such as BASIC or COBOL?

The reason is because of the main benefits of machine language which are:

* FASTER EXECUTION OF THE PROGRAM
* MORE EFFICIENT USE OF MEMORY
* SHORTER PROGRAMS (in memory)
* FREEDOM FROM THE OPERATING SYSTEM

All of the above benefits are a direct result of programming in a language that the CPU can understand without having to have it translated first. When you program in BASIC, the operating system is the machine language program that is really being run by the machine. The program is something like:

```
Next        Look at next instruction
            Translate it into a series of
               machine language instructions
            Perform each instruction
            Store the result if required
            Go to Next again
```

If you are wondering where the computer finds this program, the operating system, it is in the ROM. In other words, it is built into the Spectrum. (ROM is the abbreviation for Read Only Memory, memory locations whose content you cannot change, but can only be read/PEEKed.)

Programming in BASIC can be up to 60 times slower than a program written directly in machine language!

This is because translation takes time, and also the resulting machine language instructions generated usually are less efficient. Similarly, it is usually faster to drive yourself than to take public transport; you can take shortcuts you know, instead of following the public transport route which needs to cater for the GENERAL public CASE.

Nonetheless, we would have to be among the first to admit that programming in machine language does have drawbacks.

The main disadvantage of machine language are:

* PROGRAMS ARE DIFFICULT TO READ AND DEBUG
* IMPOSSIBLE TO ADAPT TO OTHER COMPUTERS
* LONGER PROGRAMS (in instructions)
* ARITHMETIC CALCULATIONS DIFFICULT

This means that you must make a very conscious decision of which programming method you should use for each particular application.

A very long program for financial applications should be written in a language designed to deal with numbers and one in which programs can be easily modified if required.

On the other hand there is nothing quite so bad as an arcade game written in BASIC - when you get down to it, it is just too slow.

Your own needs, the amount of memory in your computer, the response time required, the time available for development, and so on will determine your choice of programming language.

Thus, in summary, machine language is a series of commands which the CPU can understand and which can be represented by numbers.

What is Assembly Language?

Quite obviously if machine language could only be represented by numbers, very few people would be able to write programs in machine language.

After all, who could make sense of a program which looked like

```
0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
etc ...
```

Fortunately, we can invent a series of names for each of these numbers. Assembly Language is just such a representation of Machine Language so that it can be read by humans in a form that is easier to understand than

0 1 1 1 0 1 1 1.

There is only one difference between Assembly Language and Machine Language: Assembly Language is one level higher than Machine Language. It is more easily read by humans than Machine Language, but on the other hand, computers can't read Assembly Language.

It is not an adaptation of Machine Language such as BASIC. For each Assembly Language instruction there is an identical (in function) Machine Language instruction, and vise versa, ie. there is a ONE-TO-ONE relationship between them. We can therefore say that Assembly Language is EQUIVALENT to Machine Language.

Assembly Language makes use of mnemonics (or abbreviations) to enhance readability. For example at this stage, the instruction

```
INC     HL
```

may not mean much to you, but at least you can read it. If you were told that 'INC' is a standard abbreviation (or mnemonic) for INCrease and that HL is a 'variable', then by simply looking at that instruction you can get a feel for what is happening.

The same instruction in Machine Language is

0 0 1 0 0 0 1 1

Now obviously you can also "read" that instruction in the sense that you can read the number, but it isn't going to mean much to you unless you have a table to look up or when your brain is functioning almost like a computer.

Assembly Language can be converted directly to machine code by a program or by you. Such a program is called an ASSEMBLER. You can see this as a program which performs the rather boring task of translating your assembly language program into a sequence of machine language instructions that the Spectrum will understand. And we understand that an ASSEMBLER for the ZX Spectrum is already available.

Nonetheless, such assemblers typically require 6K of memory, and are therefore of limited use on a 16K machine. The Spectrum display takes up 7K of memory, and after loading the Assembler you may have only 4K of memory left for your assembly language program. (This will means about 1/2 K of machine language program).

The alternative to an Assembler program is for you to do the translation of the assembly language mnemonics into machine language by hand, using the tables provided in this book.

It's hard, it's frustrating at first, it's inconvenient, but it's wonderful practice and gives you a great insight into the way the Spectrum CPU works.

We would in fact recommend that you try writing short machine language programs in this way - ie writing them in assembly language and translating it into machine language by hand - before buying an Assembler program.

CPU

The central processing unit of the computer. This is the chip that does the controlling and calculating work in the compute

Machine Language

The language understood by the CPU. For the Spectrum's CPU, this is the Z80 machine language which is made up of about 200 instructions'.

BASIC Langua

A computer language designed to be intelligible to humans. When

but easier to write

Assembly Language

The human shorthand representation of the machine languag nstructions so that each of the latter instructions can be understood more easily. For example, HALT is the assembly language equivalent of the machine language instruction 0 1 1 1 0 1 1 0

A program that translates assembly language instructions (ly read and und rstood by human) into machine la instructions (which can be understood by the computer eg the Spectrum).

Read Only Memory (ROM

program that has been FIRMLY built into the hardware of the computer; it will remain there even when the power is off. For the Spectrum, the ROM is in Z80 machine code, and was written pecifically for it. The ROM of Spectrum occupies from memory locations 0 to 16383. You can only refer to the contents of these ocations, unlike the rest of memory which you can refer to and change as desired

BASIC Machine Language Concepts

WHAT IS THE CPU?

machine (the CPU) talks

Unless we know what sort of information the CPU understands we

do things all the time.

Especially calculatio

to keep track of what is happening. How does he do it?

The design of the CPU

only, but he is able to do those tasks very quickly.

We mentioned above that the CPU do n't even have pencil and p and that is art of the design of the CPU. Any number he can't remember or track of has to go in a box for safe keeping.

time in NEW YORK, knowing the time in LONDON.

it to do next, so it puts that information away in a box, say box #1.

result away, say in box #3.

10 - 5 = 5

The answer of course is 5 o'clock.

[illegible] so it does exactly what you or I would do — it counts on its fingers and toes.

The CPU's hands and feet are called Registers.

[illegible] hands and toes — but we will get to that later.

[illegible] the above exercise. let's call one of the CPU's hands "HAND A". How [illegible] actually do given the above instructions

* Count out the value of box #1 on the fingers of Hand A,

* Subtract the contents of box #2 from what he has already on his fing[illegible]

  Look at the value on the fingers of Hand A and store it in box #1.

[illegible] conclusions to be drawn from this

1. The CPU would not be able to deal with a number like 11.53 — it could only deal in whole numbers.

2. The CPU would be limited in its calculations to whatever number it could count on its fingers.

This is indeed true.

The main consolation however is that the CPU has a lot of han[illegible] and feet and can count on each of them separately, and that it can count to 255 using only the 8 fingers of Hand A.

[illegible] 255 and each foot can be used to count to over 64,000!

[illegible]

[illegible] done is describe the processes.

[illegible] instruct the CPU at each step

```
SETTING UP
        LD      (BOX #1),   10      ;Load box 1 with 10
        LD      (BOX #2),   5       ;load box 2 with 5

CALCULATIONS
        LD      A,  (BOX #1)        ;load A with box 1 contents
        SUB     A,  (BOX #2)        ;subtract contents of box 2

STORING THE RESULT
        LD      (BOX #3),   A       ;load box 3 with A value
```

These instructions may seem a little [illegible] at first, but after all, mnemonics are mnemonics.
"LD" is an abbreviation for LOAD so that

```
        LD  A,1
```

for example, would mean load A with 1: that is count off 'one' on the fingers of hand A.

[illegible] CONTENTS OF WHATEVER IS INSIDE THE BRACKETS.

[illegible] brackets do look like they are meant to indicate a container.

[illegible] #1 and #2 with 10 and 5, ...etc... to get the final result of 5 in box #3.

All of this is fairly simple to follow and I am sure you can [illegible]

[illegible] on Hand "A" are used to represent the time in New York. A minute later they may be used to represent the number of employees in a company, and at some other time how much money you have.

If you are used to the concept of variables from your BASIC programming, you must leave that behind in machine language programming.

The fingers of Hand "A" are not a variable in the same sense as in a BASIC program. They are merely what the CPU uses to count with.

ONE OF THE BIG DIFFERENCES IN PROGRAMMING IN MACHINE LANGUAGE AND PROGRAMMING IN BASIC IS THIS LACK OF VARIABLES

You may realise that you can think of the BOXES we use to store

Yes, you are absolutely correct, but these are not variables either. They can be immensely useful and perform similar sro purposes to variables, but bear in mind that these boxes are no more than memory locations set aside for a specific purpose.

The way the CPU copes with negative numbers is different, and we will look into that later.

What if the CPU runs out of hands?

I should mention here that you would probably find the CPU a very strange looking fellow were you to meet him in the street.

His hands have eight fingers each, and he h eight hands! He only has two feet, but each foot has 16 toes, a he is extremely agile with his toes!

and toe

Nonetheless, it is possible that in some cases the CPU will not

on or other the programmer will wish the CPU to stop in the middle of calculation to do something else

boxes it put the information in

The Z80 CPU gets away with using a stack, which is one of the

bills, spare notes, etc. I am sure you have seen those stacks where you spike one piece of paper on, and then the next one, a so on. It's a great filing system if you want the top piece of paper only, but very inconvenient if you want one in the midd because you have to riffle through all the pieces of paper on the stack

As it happens, it's a very convenient system for the CPU because it ever only needs to look at the top piece of information

Whenever an interruption causes the CPU to stop doing its calculations, it PUSHes all the information it has on its hands onto the STACK, and as soon as the interruption is over, it POPs the top bits off, and continues with its work.

[illegible]
[illegible]
get it off, we "POP" it off.

[illegible]
information, there then needs to be many separate "POP"s.

[illegible]
keep the stack stuck to the ceiling. This means that the more
[illegible]
downwards.

[illegible]
information is in - it knows it is the last piece of information
"PUSH"ed on the stack. Naturally it needs to be a little bit
[illegible]
[illegible]

What can the CPU do?

I think it's worth considering at this stage the type of
[illegible]
built into the Z80 chip.

Because the CPU has to be able to keep track of all its
[illegible]
numbers the CPU can deal with
* one handed numbers - ie numbers you can count on one hand
* two handed numbers - ie numbers you can count off on two hand

You may find this difficult to believe, but the CPU cannot deal with numbers larger than those it can count on two hands!

The types of instructions the CPU can perform are also very limited
* counting off numbers on one hand
* counting off numbers on two hands
* adding, subtracting, increasing, decreasing, or comparing one handed numbers
* adding, subtracting, increasing or decreasing two handed numbers
* various manipulations on one handed numbers - eg making the number negative
* making the CPU skip to another part of the prog
* trying to communicate one handed numbers to and from the outside world.

I am sure you will agree that this is a very limited set of
[illegible]
get the CPU to play chess, or to work out your wages!

[illegible]
have to write a program to do so

[illegible]
than writing programs in BASIC - you can only do things in tiny little steps.

S MMARY

Registers

[illegible]

[illegible]

these can be thought of as the CPU's feet. Each 'hand' has eight 'fingers', while each 'foot' has 16 'fingers'.

Memory Locations

The CPU can transfer information from its hands into or from any other other hand, and into or from memory.

[illegible] represent specific information.

The Stack

The CPU can use the stack to transfer information the programmer may wish to store temporarily. Information is [illegible] retrieved by POPping the information off.

Possible Instructions

[illegible] lest type of information transfer and simple arithmetic calculations. All programs must be made up of series of these simple instruction

The Way Computers Count

only manage to count to 10?

as having your little finger raised?

different numbers in this way.

With only two fingers it is po [illegible] ible to devise a way to count from 0 to 3, as follow

00 = 0

We can indicate not having a finger raised as '0'

[illegible]

representation 11 (or two fingers) have the value 3.

There is a direct relationship between this and binary

[illegible]

can be made to indicate on an off (or '0' and '1' as convention dictates).

[illegible]

from 0 - 7?

[illegible]

out all the possibilities for four fingers being rais [illegible]

confusion in trying to write down the number eleven as opposed to indicating that two bits were set, a universal convention has been adopted

The numbers 10 - 15 are indicated by the letters A

Decimal	10	A
	11	B
	12 =	C
	13	D
	14	E
	15	F

This means we write the numbers from 0 to 15 decimal as

Simple, isn't it?

This way of treating numbers is called the HEXADECIMAL FORMAT.

To prevent confusion, some people write "H" after a hexadecimal
10H). The "H" has no value, but serves to remind the hexadecimal convention.

In machine language programming, it is CONVENIENT to deal with numbers in hexadecimal format.

your instruction in normal decimal format, it is convenient for us to use the hexadecimal format because

1. It is easy to convert from this form to binary, which tells us which bit (or finger) is doing what.

2. It gives us an easy means of seeing whether numbers are one handed or two handed - ie 8-bit or 16 bit

3. It standardises all numbers to sets of 2-digit numbers. (We will elaborate on this

4. It is the common convention and familiarity with hexadecimal will allow you to read other books and manuals more easily

5. As the CPU is designed to process information represented by binary numbers which are cubersome for humans to read, we need a representation which is more easily readable

But it is only a convention and not a sacred rule.

T [illegible]

be represented by two hands of 5 fingers each.)

REGISTERS IS THAT THIS IS THE STRUCTURE OF THE ZX SPECTRUM

on each hand.

Taking th... one step at a time, let us ...come familiar with 4 fingers first:

left. If we number the fingers:

3 2 1 0

...hen the value of each fin...r is 2 to the power N' where N i... finger number. Let's call ... 4 finger hand a "handlet" (just ... small cigar is a cigarette

Exercise

bits (or fingers) represent?

	Decimal	Hexadecimal
00 0		
11.0		
001		
010		
100		

ast few pages again before going on.

Say 16? We would use the next finger on the left, as

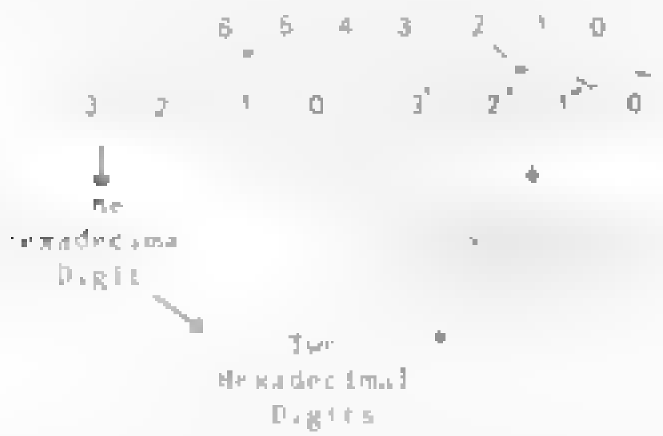
7 8 5 4 3 2 1 0

= 16 decimal = 10H (Hexadecimal)

The reason we write the number as 10H is that we divide the hand into two "4-bit handlets". We can therefore easily denote each

& A-F).

In this way any 8-bit hand can be written as exactly two hexadecimal handlets



The "handlet" on the left indicates 16 times as much as the "handlet" on the right. This is much the same way as in decimal notation, the digit in the "tens" column is worth ten times as much as the digit in the "ones" column.

We convert numbers in decimal format such as 15 automatically to
    15 = (1*10) + 5
This is so automatic that we don't even think about it.

It is exactly the same thing in hexadecimal notation. To convert

hexadecimal number on the left "handlet" by 16. Using the example above

    10H = (1*16) + 0
        = 16 Decimal

maximum is obtained when all fingers are held up

7 6 5 4 3 2 1 0

FFH = (F*16
= ((15*16) + 15 (in decimal)
= 255 (Decimal)

The smallest number is when no fingers are held up

00H = 0 Decimal

Note that all numbers, from the smallest to the larg
and only 2 digits to define the number.

soon get the hang of it.

Also that when you count in hexadecimal, you do the same
decimal

Decimal: 26 27 28 29 30 etc.

Hexadecimal. 26 27 28 29 2A 2B 2C 2D
2E 2F 30 etc

The values of the numbers in the decimal and hexadecimal series

2AH, not 30H

The following BASIC program will enable you to input to your Spectrum a decimal number and convert it to a hexadecimal value.

```
100 REM decimal to hexadecimal conversion
110 PRINT "Please input decimal value."
120 INPUT n : PRINT n
130 LET S$ = "": 135 LET n2 = INT (n/16)
140 LET n1 = INT (n - n2*16
150 LET S$ = CHR$ ((n1 <= 9) * (n1 + 48) +
        (n1 >9)*(55 + n1)) + S$
160 IF n2 = 0 THEN PRINT : PRINT "HEXADECIMAL =      S
                   = 1 TO 200  NEXT 1: RUN
170 LET n = n2: GO TO 135
```

and use the BASIC program to test your answer.

i. 16384 memory address of the start of Spectrum display file

ii. 22528 memory address of the start of Spectrum attribute file

iii. 15360 memory address of the start of Spectrum character set

iv. 15616 address of the start of ASCII characters in Spectrum

SUMMARY

Decimal

The decimal notation is a convention of counting numbers in groups of ten units at a time. These are represented by 0, 1, 2, 4, 5, 6, 7, 8, and 9

Hexadecimal

The hexadecimal notation is a convention of counting numbers in

4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Sometimes an H is added at the end of a hexadecimal number to remind us it is written in this format. For example, 1B00H

8 Bit Memory Locations

format as a two digit number.

How Information is Represented

[illegible] In a computer is stored as groups of bits.

A bit stands for Binary digIT ("0" or "1"); in the Z80A [illegible] group of eight bits is called a BYTE.

[illegible] Z80 and most microcomputer CPUs talk.

[illegible] or alphanumeric text. We will thus discuss below these thr[illegible]

Program Representation

[illegible] "sub tasks"

[illegible] two or more bytes.

[illegible] with one byte at a time, and if it requires more than one, it [illegible] instructions where possible.

[illegible]

Numeric Data Representation

* Integer Representation

[illegible]

numbers. Also, by using only 8 fingers (ie an 8-bit number), we could represent all the numbers in the range 0 to 255.

e.g. decimal 255 is represented by OFF[illegible]
the binary [illegible]

But what about negative numbers?

* Signed Integer Representation

[illegible] number one byte is a HAND with eight fingers and a number is represented by holding different fingers up.

[illegible]
following convention (signed representation):

[illegible]
highest bit - bit 7 is on.)

[illegible]
positive (depending on whether the thumb is up or not).

[illegible]

Now comes the crunch. When is a number with the thumb up a positive number and when is it a negative number?

[illegible]
particular time.

[illegible]

or positive and negative.

* Choosing a Representation for negative numbers

number is negative, and not holding the thumb up means it is positive. Is this enough?

No. We need to decide which of the 127 possibilities of the remaining 7 fingers will denote -1, which one -2, and so on.

We need a representation of negative numbers, such that when a number is added to its negative we get zero. As an exercise, let's

will be up)

could it be-)? 1

0 0 0 0   0 0 0 0

Let's try 1 0 0 0 0 0 0 1 - in other words, the same as +1

0, and convert it to zeros all along.

number which will give us the right answer is

1 1 1 1   1 1 1 1   (FFH in hexadecimal)

To confirm this

(carry)   0 0 0 0 0 0 0 0

Let's try this rule on another number, such as 3, say

(FDH

Let's add to this number to 3 and see what happens:

0 0 0 0 0 0 1 1

(carry) 0 0 0 0 0 0 0 0 It works!

We have found a way to represent negative numbers!

-01 => FF

The largest positive number is

0 1 1 1 1 1 1 1 = 7F => 127 Decimal

and the negative of this is

1 0 0 0 0 0 0 1 = 81 => - 127 Decimal

The real test of this rule is to see if by applying the rule to a negative number we get back the positive again!

Let's try it out on -3 which we worked out above is FDH.

Number 1 1 1 1 1 1 0 1
Opposite 0 0 0 0 0 0 1 0
Add 1 0 0 0 0 0 0 1 1 => 3

This is therefore a representation that works! We can apply it to get the negative of any number.

16 - bit Negatives

Exactly the same reasoning applies to two hand numbers (16-bit numbers), except that the thumb of only one hand needs to be shown as 'ON' to indicate if the number is negative or not. (ie bit 7 of the high byte).

Convention

The computer terminology for this convention is called TWO's

numbers at the Appendix of this book.

Remember that this is only a convention! You still have to decide at all times whether the numbers you are using are meant to designate numbers in the range 0 to 255 or numbers in the range -128 to +127.

Exercise

1 If 127 (0 1 1 1 1 1 1 1) is the highest positive

number which can be represented in this convention. how would you represent -128?

ii. Find the highest positive 16-bits (TWO HANDS/BYTES) the highest 16-bits negative number? iii. Find the 2's complement of the smallest 16-bit negative number 8000H. Why is it 8000H?

Alphanumeric Data Representation

perhaps

Our convention to rep pretty straightforward: all cha represented on a single hand (ie in an eight-bit code).

characters representation: The ASCII Code, and the EBCDIC Code

ASCII stands for 'American Standard Code for Information

EBCDIC is a variation of ASCII used by IBM.

You can find an ASCII conversion Table in the Appendix. Compare it

pp 183-186.

Try this : PRINT CHR$ 33
and you will get a "!", because "!" is
represented internally by 21H.

variety of things
It could be - a program instruction to the CPU
- a number in the range 0 to 255
- a number in the range - 128 to + 127
part of a two handed number
- an alphanumeric character

This is all true, and it is up to you, the programmer, to remember just what it is the CPU's hand is supposed to be holdin

SUMMARY

Memory Contents

we desire. There is no way of telling which is which just by examining the contents of a single memory location.

Programs

Program instructions are stored in memory as sequences of bytes. Some instructions require only one byte, while others require up to four bytes.

Numbers

Each memory location can be used to store either positive

from 0 to 255 or -128 to +127.

Negative Numbers

A convention has been adopted that when we choose to have memory store a signed (+ or −) number, the following rule shall apply.

If bit 7 is on, the number is negative
If bit 7 is not on, the number is positive

To obtain the negative of any number, get the "2's complement" and add 1.

2's Complement

The 2's complement of any number is its opposite in binary form. Any bit that is on becomes off, and vice ver

A Look Into the CPU

Introduction

[illegible]
under a licence from Zilog Inc.

[illegible]
0.000000286 of a second

[illegible]
that even if all instructions performed are the slowest one ... bout
160,000 instructions can be performed [illegible]

A Physical View of The Brain

[illegible] or in the Spectrum is a silicon chip with forty pins [illegible] from 1 to 40. These pins are the communication lines [illegible]
[illegible] ets its clock signals from pin 6, [illegible]
through pins 7 to 15 except pin 11. The rest of pins a[illegible]
control signals communication

[illegible] find yourself totally lost at this stage. But no me[illegible] it's really to our advantage that we don't know the [illegible] structure of the machine, and we don't need to know it to [illegible] capabilities. It's just the same as with a calculator. The [illegible]
other words we don't see it'). We are only interested in the [illegible]
and how we can use it to our purp[illegible]

Logical View of The Brain

Logically, the Z80 can be divided into five functional par

They are i. the CONTROL UNIT
ii. the INSTRUCTION REGISTER
iii. the PROGRAM COUNTER
iv. the ARITHMETIC LOGIC UNIT
v. the 24 USER REGISTERS (the usable HANDS and FEET of the CPU)

* CONTROL UNIT

We can see the CONTROL UNIT as a supervisor for the CPU's processing. Its task is to time and coordinate the input, [illegible] asked to perform, whether the instructions come from the ROM program, or from your progr[illegible]

* INSTRUCTION REGISTER

This is a HAND that the CPU uses to hold the current instruction that it is going to perform. The whole task which comprises a [illegible] de somewhere in memory - either in the ROM or in [illegible]

* PROGRAM COUNTER

[illegible] ocation of the next instruction to fetch out

* ARITHMETIC and LOGIC UNIT

This is the calculator inside the CPU. It can perform both [illegible] addition and subtraction, incrementation (adding 1) and [illegible] fingers up or down, etc.

[illegible] FLAG register. This is discussed in more detail further on.

* sc8-R SI*s

These are the CPU's Hands and Feet, which you, the programmer, can control.

There are twenty four User registers within the Z80 microprocessor - some are HANDS, and some are FEET

The images we have been building up of hands, feet and boxes make the processes easy to visualise and are a good representation of what is going on, but computer buffs tend to look askance if you say things like "...and then the computer shifted its information from its right hand to its left hand.

so that when faced with that situation, you will be able to say
"LD B,A

To start off with, computer buffs refer to the [illegible]nds and feet of the CPU as "registers"

We mentioned earlier that the CPU has eight hands these are called A, B, C, D, E, F, ...In our world, the definition of a hand is something with eight fingers.

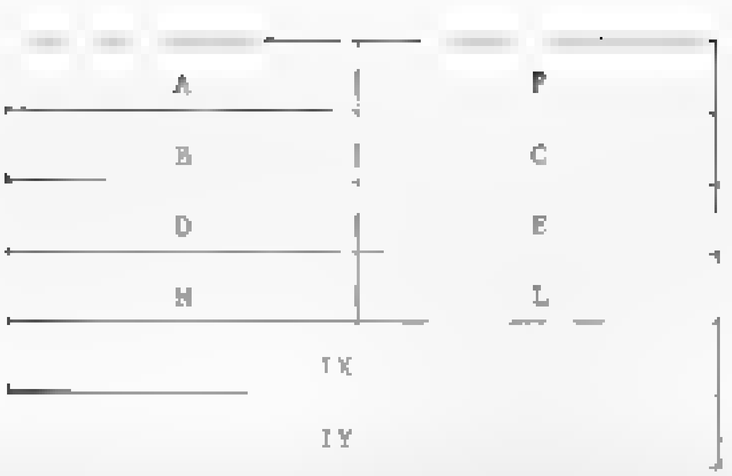
The CPU has two feet: these are named IX and IY. The definition of a foot is anything with 16 toes'

register has only one letter in its name then it must be a hand (that is, contains 8 bits), while if it has two letters in its name then it must be a foot (that is, have 16 bits).

We will have you used to computer terminology in no time.

Actually the remaining two hands for the CPU after D, E, F, ...are not named "G" and "H" as one would expect but "H" and

The conventional way to represent all these reg[illegible]ters is as follows



[illegible]

such as BC, DE, and HL.

The reason the register pair "HL" was called "HL" instead of [illegible]

[illegible] your hands and toes. You can easily set up your fingers to [illegible]

might think you meant to represent the number 73.

The "H" in "HL" stands for HIGH and the [illegible] stands for LOW, so there is no chance of confusion - right?

[illegible] register in the other register pairs contains the high number:

B in BC
D in DE

because all the highs and lows are treated in the same order

The feet (IX and IY) also have a special name: they are called "index registers". This has a lot to do with the fact that they can be used to organise information in much the same way as a book index is organised. Alternatively, you can view them as table pointers

OK [illegible] understand the term[illegible] points

THE ACCUMULATOR (A register)

[illegible]

[illegible] single byte register is the most important [illegible] when there was only a single register that could be u[illegible] 'accumulate' a result.

[illegible] A regist[illegible]

[illegible] that most people can perform some task [illegible]

The Flags:

Please note that "AF" is not usually treated as a re[illegible]ter pair. [illegible] separate chapter.

[illegible]

[illegible]

Maybe the HL register is the CPU's right foot?

An Alternate Register Set

[illegible]
also has a spare set of hands!

Not really so much a spare set of hands (all right, alternate
[illegible]
of work gloves.

[illegible]
of a hand with the number 3 counted off!

[illegible]
on the other set of gloves!

[illegible]
retained there. Nor, naturally, can the glove perform any
calculations without a hand inside the glove!

[illegible]
information the gloves retain.

[illegible]
on a right hand.

The representation of all the registers is now therefore

```
        A - F   (===)   A' - F'
        B - C   (===)   B' - C'
        D - E   (===)   D' - E'
        H - L   (===)   H' - L'
                 IX
                 IY
```

Note that the set of gloves you are wearing has the [illegible]
the hand it is for, while the spare set is always indicated with
the dash symbol.

[illegible]
only works on your HANDS, not your gloves.

The only instructions involving the alternate register set are of the "swap gloves now" type For example

```
1.  LD A, (Box #1)        ;Load A with contents of
                             x #1
2.  EX AF, AF              hort for exchange -
                           ie. swap gloves on AF
                          with those of AF'
3.  LD A,(Box #2)         ;
4.  EX AF,AF'             ; Another exchange
5.  LD A,(Box #3)         ;
```

You will note that in the above 5 instructions there are no instructions which have specifically affected the alternate

register set. Try to work out what is happening.
Do you know what will be in register 'A" after each instruction?

For simplicity's sake, let s assume that the contents of the three boxes are as follows

(Box #1) = 1
(Box #2) = 2
(Box #3) = 3

Then the following is what happens after each instruction

		Register A
1.	1	Not known
2.	Not known	1
3.	2	1
4.	1	2
5.	3	2

Really quite simple, isn't it?

or into MEMory. We will follow through this point

Even More Registers?

using these to any great extent.

The STACK POINTER

The STACK POINTER is another foot the CPU has (2-byte addressing register).

It always points to where the pile on the stack has got to. As the stack grows, it grows downward from high memory locations [illegible] memory locations.

[illegible]

it every time you do a PUSH or POP.

[illegible]

you PUSHed on to the stack. You can be sure that this will cause your program to "CRASH".

The I Register

[illegible]

base address of a table of addresses for handling different responses to an interrupt, for example, input/Output request

However in the SPECTRUM this facility is not us[illegible] register is involved in generating T.V frame [illegible] unlikely you will ever have to use this registe[illegible]

The R Regis[illegible]

The R register is the memory-refresh register. It is provided in [illegible]

[illegible]

will disappear!

[illegible]

the R register thus cycles over and over from 0 to 255.

[illegible]

without ever worrying about refreshes, etc

[illegible] hardware and system [illegible] random number [illegible]

255. We will demonstrate this usage later.

SUMMARY

User's Registers

There are eight main 8 bit registers in the CPU (A, F, B, C, D, E, H, L), and two 16-bit registers (IX and IY). Eight bit registers have only one letter in their name, while 16-bit registers have two letters.

Register Pairs

Six of the eight 8-bit registers can in some circumstances be used in pairs to operate on 16-bit numbers.

These are the BC, DE and HL register pairs. The name HL can serve to remind us which is the High order byte and which the Low order byte.

Preferred Registers

The Z80 CPU is designed in such a way that some 8-bit instructions can only be performed by the A register, while some 16-bit instructions can only be performed by the HL register pair.

Alternate Register Set

The eight main 8 bit registers can be swapped with another 'alternate' set of registers.

The values stored in the main registers are retained by the CPU while the alternate set is being used, but cannot be accessed.

Changing the register sets again allows us to operate on original values again.

This is all very well.

[illegible]

notation, and it all seems so irrelevant. It doesn't explain how you actually RUN a machine language program

[illegible]

the time! (When it's on). It's just that you are not aware of it. Even when you're not doing anything, just watching the screen, trying to think of what to enter as the first line of your [illegible] under the control of a machine language program.

This program is the one that is stored in the ROM chip and is referred to as 'the operating system'. For example, the part of the program that is running when you're sitting there looking at the screen does the following things

Scan the keyboard for entry
Note that no key has been pressed
Display the present screen (empty)

[illegible]

of the 'interpreter' type as we have alread[y] ex[p]lained. It looks at your next BASIC instruction, converts it to machine langua[ge], executes that part of the program, and then returns to interpret the next instruction.

All this stops being true when you run your own machine language program!

Total freedom from the operating system. The use of the 'USR' function hands over total control of the CPU to whatever commands you have placed at the USR address. It will interpret whatever it finds there as valid machine language instructions.

This can be pretty terrifying as you could lose everything stored in memory should you lose control. One error, one wrong character, and you will have to turn the Spectrum off and start again from the beginning.

There are no error messages to catch what you have done wrong, no syntax checking for incorrect statements - so if you make the slightest error, the hours of work you put in to enter your progra[m] could be lost!

At the end of this book we have included a BASIC program which will allow you to enter and edit machine language programs. Once you have entered this program on your Spectrum, save it on tape as it

language program at least once.

The worse that can happen is that you may have to turn your Spectrum off and on again.

Editor" found at the back of this book and RUN it.

32000. Enter the number 32000 then press (ENTER).

The screen will now show

Command or Line (###):

new line of machine code.

screen should now show you all the lines you have ente

1 c9

and at the bottom of the screen the prompt
Command or Line (###):

enter a command instead.

you have specified, namely 32000.

Congratulations: you have just entered a one instruction

ting address. Enter 32000 then (ENTER).

key "m" to return to the main command input stage.

What the instruction "C9" means is 'RETURN'

[illegible] way you want to "return" to the safety of earth (or operating system as the case may be).

[illegible] followed by (ENTER).

[illegible] used at the start.

[illegible] be the value of the BC register pair.

[illegible] same one) deals with the "USR" function.

[illegible] case 32000.

The value of "USR", as in

```
    Let A = USR 32000
```

naturally gave the answer 32000!

[illegible] running of a machine language program.

Let us enter the following machine lang[illegible]

```
0B
C9
```

[illegible] the command "dump" and then the command "run".

This time the result will be 31999! This is because the [illegible] by 1).

instructions at the table at the back. Can you work out what the abbreviations mean?

never return.

been damaged. Just turn off the power and reload everything.

Exercise

interesting.

How the CPU uses its Limbs

[illegible]

[illegible]
your Spectrum.

Imagine for a moment that you are the CPU

[illegible]
at with your other hand. There are also certain actions which
[illegible]
left and right hands.

It's the same in machine language - you can perform some tasks
[illegible]
allowed is the key to success.

The equivalent hand on the CPU to your right hand is the
[illegible]

[illegible]
hand, foot and vice ver

Computer boffins refer to this [illegible]

[illegible]
one register to another.

```
Other examples would be LD  A, B
                        LD  H, E
```

and so on

[illegible]

Thus we would read

```
            LD A, B
as          'LOAD A WITH B'
```

[illegible]
normal English sentence would be

There are also other combinations or ways other than register

another or from register to memory.

ys you can use the CPU's limb

and the possible combinations (addressing modes) that are

Let's look at the combinations offered by the Z80

Immediate addressing
* Register addressing
* Register indirect addressing
* Extended
* Indexed addressing

What a list of names? Don't worry, just remain confident and we will step through them one at a tim

The list above does not cover all the possible combinations possible - only those that apply to one-handed number

LD r, n

(or other instruction - we use LD as n example
We use the abbreviation 'r' to mean any 8-bit register and 'n' any 8-bit number.

ing is a techn que that nvolves only a single

CPU can execute the instruction IMMEDIATELY it receives the instruction. It doesn't need to look in memory to find more information in order to perform this instruction

For example, count off 215 on hand 'A'. I am sure you know enough about the mnemonics by now to be able to write this a

LD A, 215 or LD A, 0D7H

Once again you can do this with any of the registers, with any numbers whatsoever

The format for the immediate addressing type of instruction is shown below

byte 1 instruction (telling the computer what
code is this instruction)

byte 2   n   (the value of the actual
data for the instructio

Since there is one byte allocated for the actual data, the limitation to the size of number you can specify is within the range 0 - 255. If you don't understand this, refer back to chapter on "The Way Computers Count".

We usually use immediate addressing to initialise counters and to define constants needed in calculations

Immediate add essing is easy to use in machine language

would be
LET A = 5

entire programs this wa

Immediate addressing is convenient but does not solve any major problems.

But at least we're starting to get someplace! We programmers can now specify which number gets l ed onto which registers.

* Register dressing

We dealt with this mode briefly earlier. The general format is
LD r, r
(or other instructions)

This technique only involves two hands; in short, this is passing information from one hand to another.

the "F" hand (which we should not think of as a hand at all. It is the 'FLAG' register and does not store numbers in the normal sense).

Register dress instructions only need one byte.

Instructions of this type are not only short (One byte), they are faster as well. The time needed to execute them is the time taken for 4 clock pulses, or less than 1 microsecond on the Spectrum

used when possible to improve program efficiency in time and storage.

* Register indirect addressing

```
LD  (rr),   A    or      LD  A,  (rr)
LD  (HL),   n
```

This powerful type of instruction causes the transfer of data between the CPU and a memory location pointed to by the conten one of the 16-bit register pairs (Feet).

Register indirect addressing is faster than ordinary indirect

However, we must load the register originally, and so regis

same or neighboring address many times.

```
For example,    LD  HL,SHAPE    ;load HL with start of
                                 shape datab
        LOOP    LD  A,(HL)      ;Retrieve a data
                INC HL          ;move pointer along
                continue LOOP
                    until shape finishe
```

* Extended Addressing

```
LD  A,  (nn)    or      LD  (nn),A
```

Now we are looking at how to store and restore information from a to your Hand and Feet from memory.

In Extended addre ing, the instruction from the program supply the CPU w th   address specified by two bytes.

If the transaction is to and from the accumulator, nformation transfer will only affect the content of memory re rred to by the two-byte integer.

memory location will be affected

The format of this type of instruction is

byte 1	op code
byte 2	(possible additional op code)
byte 3	low order value of the 16-bits integer value
byte 4	high order value of that integer value

Now this is the way the program can read the memory into the us

referring to is relocatable

```
eg. SHAPE   DB      n,n,n,....    ;shape data b

            LD      A,(SHAPE)     ;load first byte of shape
                                   in accumulator
```

* Indexed addressing

```
LD  r, (IX/IY + d)     or      LD  (IX IY + d),    r
            (or other instruct ons)
```

This type of transaction involves a Foot of the CPU, the IX or IY index register

The CPU adds the contents of the index register to the address supplied with the instruction in order to find the effective addre

Another common 16 bit instruction type is the Block Load instructions eg. LDIR (Load increment and repeat).

One typical usage of this type of addr ing technique is to perform Table operations.

of da a. A displacement value is supplied in the instruction to

want to refer to

```
eg.     LD  IX, TABLESTART      ;initialise pointer to
                                ;start of table
        LD  A,  (IX + 3)        ;refer to the third byte
                                ;from the start of the
                                 table
```

he format of instructions of this type is

```
        byte 1      (op code)
        byte 2      (op code)
        byte 3      d                ;displacement integer d
```

The number "d" is an 8 bit number which has to be specified together with the instruction and can not be a variable. e. the range of addressing is limited from -128 to 127 from the

address pointed to by the index register.

Indexed addressing is slower because the CPU must perform an addition in order to obtain the effective address. Yet indexed addressing is much more flexible since the same instruction can handle all the elements in an array or table.

SUMMARY

[illegible] transfer it from 8-bit registers to memory.

Immediate addressing
   Defining in the program the number to be
   transferred to any register.
Register addressing
   From any register to any other regi
Register indirect addressing
   Either using BC or DE to specify the address, and
   A to hold the number to be transferred.
   Or using HL to specify the address and defining
   the number in the program
Extended addressing
   Specifying the address in the program and using A
   to hold the 8 bit number
Indexed addressing
   Using IX or IY to specify the start of a table in
   memory, and any register to hold the 8-bit number.
   The displacement from the start of the table must
   be specified in the program.
   The number to be transferred to memory can also be
   specified in the program if desired.

addressing modes are the only modes of transferring ormat,on to and from memory. No other combinations are

Instructions For One-Handed Loading Operations

Mnemonic	Bytes	Time Taken	Effect on C	Z	PV
LD Register, Register	1	4	–	–	–
LD Register, Number	2	7	–	–	–
LD A, (Address)	3	13	–	–	–
LD (Address), A	3	13	–	–	–
LD Register, (HL)	1	7	–	–	–
LD A, (BC)	1	7	–	–	–
LD A, (DE)	1	7	–	–	–
LD (HL), Register	1	7	–	–	–
LD (BC), A	1	7	–	–	–
LD (DE), A	1	7	–	–	–
LD Register, (IX + d)	3	19	–	–	–
LD Register, (IY + d)	3	19	–	–	–
LD (IX + d), Register	3	19	–	–	–
LD (IY + d), Register	3	19	–	–	–
LD (HL), Number	2	10	–	–	–
LD (IX + d),number	4	19	–	–	–
LD (IY + d),number	4	19	–	–	–

Flags notation

# indicates flag is altered by operation
0 indicates flag is set to 0
1 indicates flag is set to 1
– indicates flag is unaffected

Counting off Numbers on One Hand

learn how to count off numbers on one's hands

We discussed in the previous chapter some of the ways we can

register addressing.

from one register to another.
Examples are

    LD A,B
    LD A,E
      and so on.

Remember the terminology involved. 'LD' means
'with', and the mnemonic (abbreviation) instruction is
same order as an english sentence

We would thus read out loud something like

    LD A
    'load A with B'. The next example would be read as

hand. Even the seemingly stupid instruction 'LD A, A' is
permitted

A short shorthand of this is "LD r,r" where "r" represents any

OK. We now know we can shuffle information between han

information on those hands.

about the mnemonics by now to be able to write this as

    LD D, D7

(D7 is the hexadecimal representation of 215).

obvious, isn't it?).

Once again you can do this with any of the registers, with any

number you can specify with 8 bits 0 - 255.

A short shorthand of this is "LD r,n" where "r" indicates any

implies 8-bits still applies

Now we're starting to get someplace we can now specify which

from hand to hand. But we still haven t learnt how to put any of

many registers

We showed you very briefly an example of "extern addressing" when we were doing the time difference exerc

LD A, (Box #)

The general mnemonic for th s is

LD A, (nn)

of'.

Note two things about this

1. You can only do it with Register A
2. You have to supply the number of the box as a two handed
   b bit) number

The rev e instruction is also valid. Th s is one thing you will notice bout the Z80 - there is symmetry about the instruction set:

LD (nn),A

Do notice that these instructions only apply to Register "A" there are of course other instructions for the other registers but none quite as clear as this one. It's the dominant hand concept again

Let us pause here for a nanosecond and cons der what these two instructions actually mean and do for us.

In the first place, the number range that can be defined by a two handed number (nn) is from 0 - 65,535 This is 64K, and means that

64K! This means that all the memory - ROM, program, display, and free memory - have to fit within 64K. On a "16K Spectrum" there is

The "16K" refers to the RAM part only. On the "48K Spectrum", the

is available on a 48K Spectrum.

The instruction "LD A,(nn)" - which is read as "Load A with the contents of location nn" - is a very powerful instruction. It enables us to "read" the contents of any memory location, whether in ROM, or RAM

You can use this instruction to explore to your heart's desire, even to a location where there is no memory - eg to try to see wha.

You will be surprised - it is not all zeros

The reverse instruction "LD (nn),A" - which is read as "Load the contents of memory location nn with A" - will attempt to write to

limitat ons

You can't write to a location that can't store that information, such as in non-existent memory beyond the size of your system

One of the limitations of this instruction is that we have to know

examine or write into. The abbreviation "nn" means a definite number - eg. 17100 - and not a variab e

You can't use this instruction in the machine language equiv ent of a "For - Next" loop. The main use for this instruction is

"

are
eg. define 32000 = speed
32001 = height
2 h 2 = fuel left
in a lunar lander type progr

You could therefore plan a program where you got the fuel left, dec sed it, and stored the new amount of fuel back into that location. You will know at the time of writ ng your program the address of that memory locat on which serves to act as a strorhouse for that information

Let us be clear about this. Location 32002 is not a variable. It is only a memory location which you use to store information.

When writing your embly language program you would therefore write something like
LD A (fuel

machine code for this instruction you wou d replace "fuel" by the hexadecimal address of the memory location you specified.

But what if we don't know the exact address of the memory location where the information we seek is? Suppose we can only calculate

where that information is going to be? Because we need 16-bits to specify the address of any memory location, we would need to store it in a 16-bit register - this means one of the register pairs BC, DE, or HL, or one of the index registers IX or IY.

One way we can do this is to have one of the register pair contain the address of the memory location. Because the register contains the information and because we don't have the address directly we call this form of addressing register indirect addressing.

The mnemonic abbreviations for these are

LD r,(HL)
LD A,(BC)
LD A,(DE)

The English reading of these instructions is
"Load the register with the contents of the memory location pointed to by HL"
"Load A with the contents of the memory location pointed to by BC"
"Load A with the contents of the memory location pointed to by D

can load to any register - even H or L, as strange as that may sound - but that using BC or DE we can only load into the A register.

in the same way that the A register is the favoured single register.

store information into memory locations in a similar wa

LD (HL),r
LD (BC),A
LD (DE),A

direction the information flows in.

to the memory location.
The short shorthand of these instructions is

LD r,(IX + d
LD r,(IY + d)

"r" is again any register, and "d" is the "displacement"

confused - we don't mean register "D" but d = displacement)

The number "d" is a one handed number (8 bit number) which has to

This is the weakness of this particular instruction and means that

data

The symmetrical instruction is also available

    LD (IY + d),r
    LD (IY + d),r

don't worry you are unlikely to need it in your first few programs

The Z80 chip used in the Sinclair computers is nothing if not

we described above.

the number you want loaded) with external addressing (ie specifying the address to be loaded by using a register pair).

This is called - surprise, surprise - 'Immediate External Addressing'.

shorthand is therefore

    LD (HL),n

This is useful as you can directly fill a memory location without first having to load that value in a register.

"Immediate Indexed Addressing

This is of more limited use, and the abbreviated form for th instructions are

    LD (IX + d),n
    LD (IY + d),n

Using These Instructions in a Machine Language Program

Let's try to put some of these 'LD' instructions into practice.

'USR' machine language program the value of the 'USR' is the contents of BC. Let's run the following progr

the Loading address to 32000;

```
        1     0e 00
        2     c9
```

Now use the DUMP command to place this code into memory.

From now on, we will no longer be giving you such explicit

understanding into the point of the program.

be showing all of our programs as follow

```
    0E   00      LD C,0
    C9           RET
```

& Turn) and which instructions require 2 bytes, etc. (you will

The other point is that we shall try to make all our programs

It does not matter what you specify as your loading address.

or any other loading program you may design yourself

code into memory and then use the 'run' command in the EZ Code program) what would you expect the result to be?

The program sets the "c" register in the register pair BC to zero.

which is 32,000.

will be answer be   A. 0000
32000
C. 31896

[illegible]

and reread the chapter on "The Way Computers Count"

Now try running the following program

```
06  00     LD  B,0
0E  00     LD  C,0
C9         RET
```

[illegible]
registers B and C have been set to 0).

[illegible]
number, transferring to L, setting H to 0, and so on.

Exercise

[illegible]
to the attribute file by the following program

```
26 58      LD  H,58H
2E 00      LD  L,0
```

[illegible]
using the LD (HL),n command.

The structure of the attribute file is described in the Spectrum manual. Let us set the first character to ink red, paper white, flash on. This is

1 0 1 1 1 0 1 0 = BAH

so the next line of the program will read

```
36 BA      LD  (HL),BAH
```

[illegible]
program, so the last line must be

```
C9         RET
```

RUN this machine language program. Did it work?

Flags and Their Uses

Flags are those nice buntings you can wave on state occasions.......... - wrong!

[illegible] that a certain condition exists.

[illegible] indicate distress, country, piracy or whatever.

[illegible] just performed.

[illegible] at the start of the last chapter a [illegible] instructions to be discussed in that chapter, and that [illegible] chapter affected any of the flags.)

[illegible] easiest to underst[illegible] Flag.

[illegible] register is zero.

[illegible] you would only need one bit to define the zero flag.

[illegible] on or off and require only one bit.

The Different Kinds Of Flags

[illegible] could only think of 6 flags!

```
S   Z   -   H   -   P V   N   C
|   |       |       |     |   |
|   |       |       |     |   |
|   |       Half carry flag   |   |
|   |               Parity flag   |
|   |               Overflow flag |
|   |                         Subtract flag
|   |                                 Carry
|   |                                 flag
```

Actually the designers thought of seven flags, but decided that one register could serve as both flags - the parity/overflow flag.

Let us now look at each of these flags in detail.

Zero Flag

This is the flag we have already discussed above. Its application is obvious, and the flag is usually set after an arithmetic operation as it serves to indicate the contents of the 'A' register.

Note carefully however that it is possible to have the 'A' register contain 0 and for the zero flag not be set. This could easily happen by using the

LD A,0

instruction. We mentioned above that none of the one-handed (8-bit) load instructions have any effect on any of the flags. The zero flag would NOT be set yet A would contain zero.

The zero flag is also set if the result of the "rotate and shift" group of instructions results in a zero.

As well, the zero flag is the only visible result of some testing instructions, such as the "bit testing" group of instructions. In those cases the zero flag is put on if the bit tested is zero.

Sign Flag

The sign flag is very similar to the zero flag and operates on very much the same set of instructions (with the major point of departure being the "bit testing" group where the concept of a

negative bit is somewhat meaningless in any case.

Carry Flag

This is one of the more important flags available in language, for without it the results of assembly lang arithmetic would be totally meaningless

The point to remember is that embly language instructions always refer to eith one-handed 8-bit, or two-handed (16-bit) numbers.

This means that the numbers we are dealing with can be either.

0 - 255
0 - 65535

include carry,

a) 0 - 256
.) 0 - 65536

2 0
- 2.1

ult = 255

This is a direct consequence of only having a limited number range available, and the same thing can obviously happen with 16-bit number

We've already discussed that you can only count to 255 on one ha What happens if a register is already showing 255 and you add 1? You might like to think of the register as operating the same way of your car. Once you have reached the maximum, it 'clocks' over and begins counting from zero again.

In the same way, if the register or car meter shows all zeros, and you turn it backwards, you will get the highest value showing, or 255 on an 8-bit register.

This is why the result of 200 - 201 gives 255. If we were car dealers we would obviously like an indication that the meter has clocked over, whether in a forward direction - in which case the car has travelled further than it seems - or a backwards direction in which case the meter has been tampered with.

is called the carry flag. Fortunately we do not need to worry about registers being tampered with

We have seen that the carry flag can be set by subtractions if

operations if there should be an 'overflow'.

It is therefore convenient to think of the carry bit as the 9th bit of the 'A' register

Number	Carry bit	Number in bit form
142		
+ 135		
267	1	

But as we do not have 9 bits, the 'A' register would contain the number OBH (Decimal 11) and the carry would be on (ie. = 1).

leave a '1' there as well.

Using Flags in the

Machine Language Equival

such as

if A=0 then
whe what follows can be 'Let....'
or 'Goto...'
or 'Gosub..'

exact the same kind of decision can be programmed in machine langua (except for the 'Let...'). Instead of saying "If A=0", we ely look at the zero flag If it is on, then we know A=0.

the only ones which allow us to make a choice in the next instruction to be executed

The format of such instruction is as follows:
For example.

JP cc, End

where 'JP' is the mnemonic for 'jump' and 'end' is a convenient label.

The instruction is read in English as "jump on condition cc to End".

The condition "cc" could be any of:
- Zero,
- Not zero
- Positive
- Minus
- Carry set)
- No carry)

The other three flags tend not to be of so much use in every day programming. They are

Parity/Overflow Flag

This flag acts as the parity flag for some instructions, and as the overflow flag on others, but there is rarely any confusion as the two types of operations do not commonly occur together.

The parity side of it comes into effect during logical operations and is set if there is an even number of set bits in the result. We deal with this in greater detail in the chapter on logical operations.

The overflow is a warning device that tells you that the arithmetic operation you have just performed may not fit into the 8-bits. Rather than actually telling you that the result needed a 9th bit, this tells you that the 8th bit changed as a result of the operation.

In the example above, adding 132 and 135, the 8th bit was '1' prior to the addition and '0' afterwards, so that the overflow would have been set. But the overflow would also be set by adding

Subtraction Flag

This flag is set if the last operation was a subtraction

Half Carry Flag

This flag is set in a manner similar to the carry flag but only in the case of an overflow or borrow from the 5th bit instead of from the 9th bit.

Both the subtract flag and the half-carry flag are of use only in "Binary coded decimal" arithmetic, and we deal with these flags in the chapter on "BCD Arithmetic".

SUMMARY

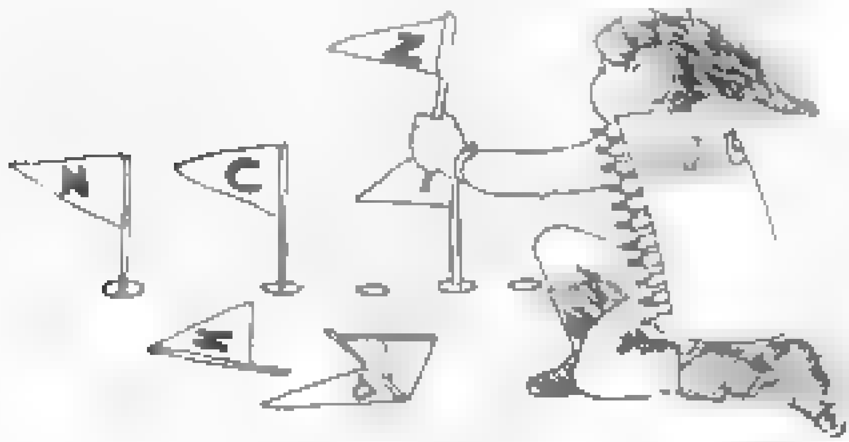
Flags are used by the CPU to indicate certain conditions after

There are six such flags, each of which can be said to be
FF. The six bits representing these flags
bits in the F register. The other two bits

The conditions indicated by the various flags are

Parity or Overflow
Sign
Negate
Half Carry

Not all instructions affect each flag. Some affect all flags, only specific flags, while others have no effect on the flags.



Counting Up and Down

numbers onto its fingers and toes

fingers or we can decrease the number represented.

essentially: whatever number you have on your fingers, incre[illegible] it by one.

monitoring the traffic at a particular intersection

Counting Up

mnemonic

```
INC r
```

INC" is read in English as 'incr[illegible] and is therefore fairly [illegible]lf-explanatory.

feet, as we saw).

This increasing of the count on our toes is written

```
INC rr
INC IX
INC IY
```

using 8-bit numbers and which are 16-bit numbers

The 8-bit numbers are denoted by a single letter, while

The 16-bit numbers are denoted by two letters.

index registers or the 'favoured register pair', HL

```
INC (IX + d)
INC (IY + d)
INC (HL)
```

(where is the displacement – not the register D')

Important not

Remember carefully our convention of reading brackets

brackets —mean→ 'contents of'

This is very important there is a lot of similarity between the instructions

```
INC HL
INC (HL)
```

but a world of diffe ce in their execution.

first would be read a cond would be read as "increase the contents of the location whos address is HL. (This second reading is often abbreviated to contents of HL.)

As long as you remember the rules of the mnemonic will be saved from this kind of confusion Let us examine how each , and let's assume that HL = 5800H.

Look at HL. Increase the count on its fingers by one Result

HL = 5801H

INC (HL): Look at HL. Find the memory locat on referred to by this number. Increase the count in that location by one. Result:

HL = 58
(5800H) (5800H) + 1

These are significantly different operations. (You might like to R N both versions – 58 H is the start of the attribute file). Note also that wh le 'INC HL' is an instruction acting on a 16-bit number, 'I C)' is an instruction which acts on an 8-bit number only – the nu ber stored in location 5800H

Decreasing the Count

certainly ensure that everything you can increase you can also decrease, and this is indeed the cas

```
DEC r
DEC rr
DEC IX
DEC IY
DEC (HL)
DEC (IX + d)
DEC (IY + d)
```

[illegible]

Effect on Flags

[illegible] very good place to review the operation of the flags

[illegible] NOT [illegible] 8 bit number [illegible]

Sign: This flag will be set (=1) if bit 7 of the 8-bit result is 1.
This means it will be on if the thumb is up using our previous analogy. Note that this will happen whichever convention you are using for the number.

Zero This flag will be set (=1) if the 8-bit result is zero.

Overflow: This flag will be set (=1) if the conten[illegible] bit 7 of the 8-bit number is changed by the operation.

[illegible] set [illegible] there [illegible] into or a borrow from bit 4 of the 8-bit number.

[illegible] is set if the last instruction was a subtraction. Thus it is not set (=0) for "INC" and set (=1) for "DEC".

[illegible]

Use the "LD", "INC" and "DEC" group of instructions to ret[illegible] numbers you want as a result of the 'USR' operation.

This will give you familiarity with these instructions.

S MMARY

ease or decrease the contents in any of the 8-bit registers or in any of the 16-bit register pairs or in either of the 16-bit indexing registers

We can also increase or decrease the contents of memory locations whose address is specified by the HL register pair or by the indexing registers

or in memory, affect all the flags except the carry flag.

Instructions for One-Handed Arithmetical Operations

Mnemonic	Bytes	Time Taken	Effect on Fl C	Z	PV			
ADD A, register			#	#	#	#	0	
ADD A, number			#	#	#	#	0	
ADD A, (HL)			#	#	#	#	0	
ADD A, (IX + d)			#	#	#	#	0	
ADD A, (IY + d			#	#	#	#	0	
ADC A, register			#	#	#	#	0	#
ADC A, number			#	#	#	#	0	#
ADC A, (HL)			#	#	#	#	0	#
ADC A, (IX + d)			#	#	#	#	0	#
ADC A, (IY + d)			#	#	#	#	0	#
SUB register			#	#	#	#		#
SUB number			#	#	#	#		#
SUB (HL)			#	#	#	#		#
SUB (IY + D			#	#	#	#		#
SUB (IY + D)			#	#	#	#		#
SBC A, register			#	#				
SBC A, number			#	#				
SBC A, (HL)			#	#				
SBC A, (IY + d			#	#				
SBC A, (IX + d			#	#				
CP register			#	#				
CP number			#	#				
CP (HL)			#	#				
CP (IX + d			#	#				
CP (IY + d			#	#				

Flags Notation

indicates flag is altered by operation
indicates flag is set to 0
indicates flag is set to 1
indicates flag is unaffected

One Handed Arithmetic

[illegible] be carried out through our dominant hand, register A.

[illegible] dominant [illegible] and knows how [illegible]

[illegible] that the abbreviation 'A' is even omitted in some mnemonics. For [illegible] normally expect to see

```
        SUB A,B
```

but in fact the mnemonic is

```
        [illegible]
```

[illegible]

[illegible]	Add any single register to A
[illegible]	Add any 8-bit number to A
[illegible]	[illegible] 8-bit number in the box whose [illegible] is given by HL
ADD A, (IX + d)	[illegible] 8-bit number in the box whose [illegible] is given by IX + d
ADD A, (IY + d)	[illegible] 8-bit number in the box whose address is given by IY + d

[illegible] location

The one that is missing is

```
        ADD A, (nn)
```

where we define the address in the course of the program.

[illegible] way to get such an [illegible] writ[illegible]

```
        LD HL,nn
        ADD A, (HL)
```

[illegible] BC or DE register [illegible]

[illegible] as we have already seen

For example, LD A,8CH

```
ADD  A, 81H
```

[illegible] A' bu[illegible] to indicate the result did not fit in.

[illegible] convert the numbers to decimal and check the addition

Hexadecimal addition and subtraction is the same [illegible] ordinary arithmetic

```
            1 = 2
            2 = 3
            u get to  1+9 you [illegible]
            9 = A
            A = B
            u get to  1+F you get
            F = 10
```

[illegible] er a number bigger than 'F' instead of '9' as in decima[illegible] arithmetic.

[illegible] be as follows

```
        +  8[illegible]

           [illegible]OH      as 8 + 8 = 16  →  10H
```

What can you do about this carry error?:

[illegible] into the Carry.

This is a very useful instruction: "ADC", which we read as "ADD WITH CARRY".

[illegible] 'A', except that the carry is added on (if it is set).

[illegible] numbers [illegible] a chaining operation

[illegible] result in [illegible]

```
        LD  A,E8H        ;Lower part of 1st no.
        ADD A,D0H        ;Lower part of 2nd no.
        LD C,A           ;Store result in C
```

```
LD A,07H        ;Higher part of 1st no.
ADC A,07H       ;higher part of 2nd no
LD B,A          ;Store result in B
```

After the first addition (E6 + D0), we will have the carry set

(check this for yourselves!)

carry

rather than in a register pair.

8-BIT SUBTRACTION

This is exactly the same as 8-bit addition. Two sets of commands exist, one for ordinary subtraction, and one for subtraction with carry

SUB s = Subtract s
SBC s = Subtract s with carry

The notation 's' is meant to denote the same range of possible operands as for the add instruction.

COMPARING TWO 8-BIT NUMBERS

[illegible] exactly what it is we mean when we compare two numbers

[illegible] same - they are 'equal'. One way to denote this in an arithmetical [illegible]

[illegible] subtracting the new number would be negative.

Similarly if the new number is smaller, then the difference would be positive.

[illegible]

```
        LD A,5          ;Number we have
        SUB N           ;Number being compared
```

Then we will have the following results -

```
If N = 5      Zero flag set, carry flag not set
If N < 5      Zero flag not set, carry not set
If N > 5      Zero flag not set, carry flag set
```

It is therefore clear that the test for equality will be the zero flag, and the test for "greater than" will be the carry flag. (The test for "less than" is the absence of both flags).

[illegible] have been altered by the operation.

[illegible] are the same as for addition

"Compare" is exactly the same as "subtract" except that the contents of 'A' are unchanged. The only effect is therefore on the flags.

Eight bit arithmetic on the Z80 is limited to

- addition
- subtraction
- comparison

and can only be performed through the A register.

Given this limitation however, a wide range of addressing modes exist

affected by arithmetical operations. We can use this as a warning of overflow.

Instructions for Logical Operators

Mnemonic	Bytes	Time Taken	C	Z	PV	S	N	H
AND Register	1	4	0	#	#	#	0	1
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	#	#	[illegible]	[illegible]
AND (HL)	1	7	0	#	#	#	0	1
AND (IX + d)	3	19	0	#	#	#	0	1
AND (IY + d)	3	19	0	#	#	#	0	1
OR Register	1	4	0	#	#	#	0	0
[illegible]	[illegible]	[illegible]	[illegible]	#	#	#	[illegible]	[illegible]
OR (HL)	1	7	0	#	#	#	0	0
OR (IX + d)	3	19	0	#	#	#	0	0
OR (IY + d)	3	19	0	#	#	#	0	0
XOR Register	1	4	0	#	#	#	0	0
XOR Number	2	7	0	#	#	#	0	0
XOR (HL)	1	7	0	#	#	#	0	0
XOR (IX + d)	3	19	0	#	#	#	0	0
XOR (IY + d)	3	19	0	#	#	#	0	0

Flags Notation

[illegible] altered by operation
[illegible] set to 0
[illegible]t to 1
[illegible] [illegible]naffecte[illegible]

Logical Operators

There [illegible] operations which are [illegible] programming as the [illegible] ordinary arithmetic.

[illegible] are:

AND
OR
XOR

[illegible] on the individual bits of the number (or fingers of the CPU's hand).

Let us look at one of these operations, 'AND':

Bit A	Bit B	Result of Bit A 'AND' Bit B
0	0	0
1	0	0
0	1	0
1	1	1

[illegible] AND [illegible] '1' only if A and B both contained a '1'.

[illegible] AND [illegible] number.

[illegible] 'What is the point [illegible] operation?'

The [illegible]

[illegible] we wish to [illegible] information, the number would be at least 8:

eg. 0 0 0 0 0 1 0 1 =5
(——————)
These bits must be '0'.

[illegible] a number whose [illegible] we do not know and apply

[illegible] the result [illegible]
lies in the range 0 – 7

```
eg.            0 1 1 0   1 0 0 1     = 105
               0 0 0 0   0 1 1 1     = 7 =) Mask

result of AND  0 0 0 0   0 0 0 1     = 1 =) in
                                     range 0 – 7
```

[illegible] place with the 'A' register. 'A' can be 'AND'ed with an 8-bit [illegible] or with (IY+d).

```
e.g.   AND 7       Note that as only the 'A' register
       AND E       can be acted on, it need not be
       AND (HL)    mentioned in the instruction.
```

[illegible] is true for the other Boolean operations, 'OR' and 'XOR'.

The 'OR' operation is very similar in concept to the 'AND' operation:

Bit A	Bit B	Bit A 'OR' Bit B
0	0	0
0	1	1
1	0	1
1	1	1

It is obvious that the result of an 'OR' operation is to give us a '1' if either A or B contained a '1'.

[illegible]

```
        LD    A,Number
        OR    1                ;make number [illegible]
```

The above two lines would be a typical assembly listing

[illegible] programming [illegible]

The result of 'XOR' is a '1' only if one of A or B contains a '1'.

[illegible] the same as [illegible]

all cases except when both A and B contain a '1'.

Bit A	Bit B	Bit A 'XOR' Bit B
0	0	0
1	0	1
0	1	1
1	1	0

have on the flags.

Zero Flag — This flag will be on (=1) if the result is zero

Sign Flag — This flag will be on (=1) if bit 7 of result if set

Carry Flag — Flag will be off (=0) after 'AND' 'OR' 'XOR' ie. carry will be reset

Parity Flag (Note that this flag also doubles overflow flag) — This flag will be on (=1) if there is even no. of bits in the result
0 1 1 0 1 1 1 0 => OFF
0 1 1 0 1 0 1 0 => ON.

Half-Carry Flag) Subtract Flag — Both flags turn off (=0) after 'AND' 'OR' 'XOR'. These flags are useful if 'BCD' arithmetic is being used.

Use of Boolean Operations on F

There is a [illegible]ecial case of the Boolean operators which is very [illegible] e of the register A operating on itself.

A is unchanged, carry flag cleared
A is unchanged, carry flag cleared
A is set to 0, carry flag cleared.

byte to do what might otherwise require two, such as LD A,0.

The carry flag often needs to be cleared - eg. as a matter of routine before using any of the arithmetic operations such as

ADC Add with carry
SBC Subtract with carry.

and this can easily be done by the instruction AND A without affecting the contents of any of the registers.

SUMMARY

There are three logical operators which are useful in machine language

AND
OR
XOR

[illegible] be stored in the A register. The result of the operation is returned in the A register.

[illegible] different to its meaning as a BASIC instruction.

[illegible] individual bits.

Coping with Two Handed Numbers

[illegible] far [illegible] we ha[illegible]e been deal[illegible]ng on[illegible]y with one[illegible]handed [illegible] numbers, but we have [illegible] about the fact that the [illegible] can [illegible] handle two-handed (16-bit) numbers in some cases.

One case we have already mentioned is the index registers. These "feet" have 16 [illegible]oes" (16 bits), and can only handle 16-bit numbers.

As we [illegible] hands [illegible] [illegible] these hands [illegible] together [illegible] "register pairs". They are BC, DE, and HL.

The [illegible] numbers [illegible] way [illegible] slow and limited.

Let [illegible] addressing modes [illegible] contortions?) available for dealing with 16-bit numbers

Immediate Extended addressing

[illegible]

LD rr, nn
(or other instruction)

This is the equivalent of 8-bit immediate addressing. It is merely [illegible] addressing [illegible] immediate [illegible] transfer.

[illegible] 16-bit [illegible]

[illegible] Extended Addressing is as follows

Byte 1	Instruction	
Byte 2	n1	Low order byte of the number
Byte 3	n2	High order byte of the number.

We use this [illegible] addressing [illegible] the [illegible] of a register pair, [illegible] example [illegible]

Register addressing

[illegible]
the registers

[illegible]
limited in the register combinations allowed.

e.g.   ADD HL, BC

[illegible]
later chapter.

Register indirect addressing

[illegible]
location is held by a register pair.

In the Z80, this ty[illegible] of addressing is again mainly applied using the register pair HL

JP (HL)

[illegible]

[illegible]ing is similar in concept to register indirect [illegible]ressing, except that the value you want is not held in a register pair, but in a pair of memory locations.

e.g.   LD HL, (nn)
where nn must be specified at the program stage.

Exercises.

[illegible] programs

1. Immediate extended addressing

```
010F00     LD  BC,15        ;load BC with value 15
C9         RET              ;return
```

[illegible] is run, you will see that the value of USR on [illegible] it.

[illegible] value of the number in the program.

2. Register addre[illegible]

We will now add a line to the program above:

```
210040     LD  HL, 4000H    ;load HL with 16384
010F00     LD  BC, 15       ;load BC with 15
 9         ADD HL, BC       ;add the two numbers
C9         RET              ;return
```

If you run this program, you will still get the [illegible] above, namely 15! Why? Didn't we add 16384?

[illegible] pair, so we didn't see any of it! To see what happened, we [illegible] add a few lin[illegible]

3. Extended [illegible]

```
[illegible]
22647D     LD  (7D64H),HL    [illegible] in 32100 and 32101
ED4B647D   LD  BC,(7D64[illegible]   ;get value of BC from
                             ;32100 and 32101
C9         RET
```

[illegible] transferring [illegible] from HL [illegible] the PUSH and POP [illegible] are more [illegible] illustrate what [illegible] addressing modes [illegible]

[illegible] command to check on this program as well.

Manipulating Numbers with Two Hands

In the earlier chapters we have seen just how [illegible] manipulating numbers in one hand [illegible] way it can handle two-handed numbers

Th[illegible] hand. Why then bother with two-handed numbers?

[illegible] 8-bit numbers [illegible] the 8-bit numbers our computer would indeed be a very limited machine

[illegible] the address of a memory location. We implied that such a [illegible] as LD A,(HL).

[illegible]

[illegible] next chapter will deal with 16-bit arithmetic.

Specifying Addresses with 16-Bit Numbers:

Please note that all addresses must be specified by a 16-bit number.

You just can't specify an address with only 8-bits, even if it is only addresses from 0 to 255. The way the CPU works, it's not an address unless it is 2 bytes of 8 bits each.

We implied this when we used the short shorthand of

```
LD A, (nn)
```

[illegible] 16-bit numbers are stored in register pa[illegible] [illegible] number first [illegible] back again with our chapter on "A Look into the CPU" . . . - "HL" stands for H = "high"; L = "low

Storing 16-Bit Numbers in Memory

There is one facet of Z80 design which is very difficult to explain or justify:

Instructions For Two-Handed Loading Operations

Mnemonic	Bytes	Time Taken	Effect on C	Z	PV	S
LD Reg pair, Number	3/4	10	–	–	–	
LD IX, Number	4	14	–	–		
LD IY, Number	4	14	–		–	
LD (Address), BC or DE	4	20	–	–	–	
LD [illegible]	[illegible]	[illegible]				
LD [illegible]	[illegible]	[illegible]				
LD [illegible]	[illegible]	[illegible]				
LD [illegible]	[illegible]	[illegible]				
LD [illegible]	[illegible]	[illegible]				
LD [illegible]	[illegible]	[illegible]				
LD [illegible]	[illegible]	[illegible]				

Flags Notation

# Indicates flag is altered by operation
0 Indicates flag is set to 0
1 Indicates flag is set to 1
– Indicates flag is unaffected

from that of register pairs is used.

The low bit is always stored first in memory!

memory

Before:		Location	Contents
		32000	00
H	L	32001	00
01	02	32002	00

memory locations are all empty.

After:		Location	Contents
		32000	02
H	L	32001	01
01	02	32002	00

listings) is that the low bit is always stored first.

live with it.

this carefully and make sure that you are

single most important source of errors in programs!

In registers: High bit stored first
In memory and programs: Low bit stored first

we have to pay.

contents of memory using the "mem" command.

Loading 16-Bit Numbers

[illegible]

LD rr, nn

[illegible] using the notation of 2 letters to indicate a [illegible] "rr" means any register pair, "nn" any 16-bit [illegible]

[illegible] tables at the back of the book - then the discussion we had on the order of the 16-bit numbers in memory becomes crucial.

Even if you do have an assembler, you should be aware of these reversals of order to enable you to "read" the code when [illegible] into memory.

Let us look at a specific example.

Load HL with 258

The mnemonic for this is

LD HL,0102H

[illegible]

[illegible] XX XX

[illegible] that the number 0102H needs to be inserted in place of [illegible] but because of the reversal rule, we do not enter this as 1002H.

The proper instruction is therefore:

21 02 01

In our examples we will show you this as

```
21 02 01          LD  HL, 0102H  ( = 258)
```

[illegible] your own programs.

Other 16-Bit Load Instructions

[illegible] 16-bit numbers [illegible]

LD IX, nn
LD IY, nn

[illegible]
location).

The general instructions are

```
        LD (nn), rr
        LD (nn), IX
        LD (nn), IY
```

[illegible]
location nn with the value in register IY .

[illegible]
into the register pair. It is not necessary to specify both
[illegible]
16-bit operations.

[illegible]
whatever is in a specific pair of memory locations

```
        LD rr, (nn)
        LD IX, (nn)
        LD IY, (nn)
```

[illegible]

[illegible]
and 23654.

In BASIC we can determine this by using the line

```
        PRINT PEEK 23653 + 256 * PEEK 23654
```

[illegible]
(23653 = 5C65H)

```
    ED  4B  65  5C        LD  BC, (23653)
    C9                    RET
```

[illegible]
THE OTHER WAY AROUND.

[illegible]

this once in each program.

[illegible]
recall, the value of USR is the contents of the BC register pair when the machine language program has finished.

Note that "LD BC, (NN)" is a four-byte instruction.

[illegible]
two byte variables listed in the Spectrum manual on pages 173 - 176.

SUMMARY

[illegible]
[illegible]
memory location where the 16 bit number is to be found.

[illegible]
the register pairs or from the index registers.

[illegible]
[illegible]
in program instructions involving 16-bit numbers)
The low byte is always stored first !!!

structions for Stack Operations

Mnemonic	Bytes	Time Taken	Effect on flags C	Z	PV	S	N	H
PUSH Reg pair	1	11	-	-	-	-		-
PUSH IX or IY	2	15	-	-	-	-	-	-
POP Reg pair	1	10	-	-	-	-	-	-
POP IX or IY	2	14	-	-	-	-	-	-
LD SP, Address	3	10	-	-	-	-	-	-
LD SP, (Address)	3	20	-	-	-	-	-	-
LD SP, HL	1	6	-	-	-	-	-	-
LD SP, IX or IY	2	10	-	-	-	-	-	

Flags Notation

\# Indicates flag is altered by operation
0 Indicates flag is set to 0
1 Indicates flag is set to 1
\- Indicates flag is unaffected

Manipulating the Stack

You may recall the image we developed in the beginning of the book of the stack as being where the CPU was able to keep information without having to remember the address of that particular information.

One of the disadvantages (possibly inadvertent) of the stack operations is that we can only PUSH and POP information in two-handed chunks. This is because the stack is primarily designed to remember addresses and we need to specify addresses as 16-bit numbers.

The general instructions for pushing information on to the stack are

```
PUSH rr
PUSH IX
PUSH IY
```

and the general instructions for popping information back from the stack are

```
POP rr
POP IX
POP IY
```

These are exceptionally simple instructions, mainly because of the lack of need to specify an address.

For the ordinary register pairs - ie. not the index registers - these instructions are only a single byte long and therefore very economical in terms of programming space.

PUSH instructions are also non-destructive - that is, the register pair remains the same information after the PUSH.

Note that because we can PUSH any register pair and POP any register pair, the register you POP need not be the same as the one you PUSHed.

For example

```
PUSH   BC
POP    HL
```

The effect of these two instructions is to leave the contents of the BC register unchanged but set the HL register to whatever the contents of the BC register was at the time of the PUSH instruction.

This effectively adds an instruction of the type

```
LD rr, rr'
```

to the 16-bit load group which was conspicuously missing.

[illegible] POP instruction for register pairs is only one byte long, the cost in terms of memory is not expensive

The other extra benefit is that we are able to PUSH or POP the register pair AF. This is one of the few instructions where AF is [illegible] obviously sensible because [illegible] preserve the contents of the flags.

[illegible] PUSH AF [illegible] POP AF [illegible] flags unchanged

Moving the Stack Around

[illegible] are PUSHed to or POPed from.

[illegible] have 16K of memory or whether you have 48K

[illegible] but not in any of the LOAD, etc. instructions because it [illegible] the [illegible]

[illegible] instruction that is available.

```
LD SP, nn
LD SP, (nn)
LD SP, IX
LD SP, IY
```

[illegible] looking in the last 30 - 40 bytes before RAMTO

[illegible] in the stack

[illegible] your [illegible] to turn the power on again [illegible]

requires on the stack and changes will cause it to bomb out.

stack pointer unless you are sure of what you are doing.

Note

miscalculation may lead to strange results

Exercis

USR subroutine is called from by 'POP'ing the value out of the

```
C1          POP BC      ;Get address in BC
C5          PUSH BC     ;Put it back on the stack
C9          RET
```

Instructions for Two Handed Arithmetic

Effect on Flags

	Z	Pv	S	
#			-	0
#			-	0
#	#	#	#	0
#	#	#	#	0
#	-		-	0
#			-	0
#			-	0
#			-	0
#			-	0
#			-	0
#	#	[illegible]		
#	#	#	#	

Flags Notation

Indicates flag is altered by operation
Indicates flag is set to 0
Indicates flag is set to 1
Indicates flag is unaffected
Indica[illegible] effect is not known

[illegible] to have [illegible] [illegible] [illegible] [illegible] negative numbers are to be permitted).

[illegible] why in some early [illegible] [illegible]

[illegible] The range of options is just not there!

Favoured Register [illegible]

In the same way that the 'A' register is the favoured register in 8 bit arithmetic, so there is [illegible] arithmetic, and it is the HL register pair.

[illegible] we do not omit the name of the register pair.

Addition

The additions are quite straightforward

```
ADD HL,BC
ADD HL,DE
ADD HL,HL
ADD HL,SP
```

But that is it!

[illegible] we need to

```
LD  DE,nn
ADD HL,DE
```

[illegible] too often.

[illegible] registers. You will also remember that there is no LOAD [illegible]

```
        PUSH IX
        POP  DE
        ADD  HL,DE
```

The one point of note is the 'SP' register - the stack pointer.

proper register, but obviously you can't use it as a variable!

the contents of 'SP' at wil

Effect on Flags

that the 'add' instruction is not a subtraction')

by the calculation.

Add With Carry:

Because of the limited nature of 16-bits, we are able to chain

'ADC' - operates in a similar manner to 'ad
range of register pairs.

```
        HL,BC
        HL,DE

        HL,SP
```

16 Bit Subtraction

to clear the carry flag before any subtraction operation.

```
        SBC   HL,BC
```

[illegible] SP [illegible] result negative will be the amount [illegible] free [illegible] simple program to do that [illegible] the chapter to confirm your solution).

Effect Of Carry Arithmetic On Flags:

[illegible]

by the simple 16-bit addition instructions

[illegible] these is set according to the result of the operation.

Index Register Arithmetic

[illegible]

registers is extremely limited:

Adding the 'BC' or 'DE' register pair
Adding the index register to itself
Adding the stack pointer.

Solution to Memory Left Exercise

[illegible] and 23654 in the Spectrum manual.

Obviously if we lo[ad] HL with the contents of that location we are halfway there

```
        LD   HL,(STKEND)
```

then subtract the 'stack pointer' (SBC HL,SP ?)

Because of the 'carry' we need to clear the carry flag. This is most easily achieved by the 'AND A' instruction, which is covered earlier in the book (p77).

```
        AND     A
        SBC     HL,SP
```

[illegible] d don't know how to do it. One-quarter marks if you forgot all about the carry.

[illegible] negative.

[illegible]

this). We first want to shift HL to BC, but there is no 'load'

[illegible]

pop

```
        PUSH HL
        POP  BC
```

HL still has the same information as before, so H [illegible]

[illegible]

again

```
        AND A
        SBC HL,BC
        SBC HL,BC
```

[illegible]

ie. the positive number of bytes left.

[illegible]

to get a result from the 'USR' function. To get HL back into BC

```
        PUSH HL
        POP BC.
```

and finally a return from the USR function:

```
        RET
```

Did you get this right?
Notice how handy the stack is

Loops and Jumps

[illegible] really getting places.

[illegible] difficult to follow, and almost impossible to debug.

[illegible] away from good program design.

Machine Language Equivalent of 'GOTO':

In BASIC, you are familiar with the instruction 'GOTO', which [illegible] you 'GOTO'.

[illegible] the next instruction and you are half-way there.

The most simple instruction is "Jump To":

```
JP  XX XX
JP  (HL)
JP  (IX)
JP  (IY
```

[illegible] status of one of the flags, such as the carry flag. This conditional jump instruction is

```
JP  cc, nn
```

where cc is the condition to be met. If we had

```
JP  Z,0000
```

[illegible] and start again with 'K').

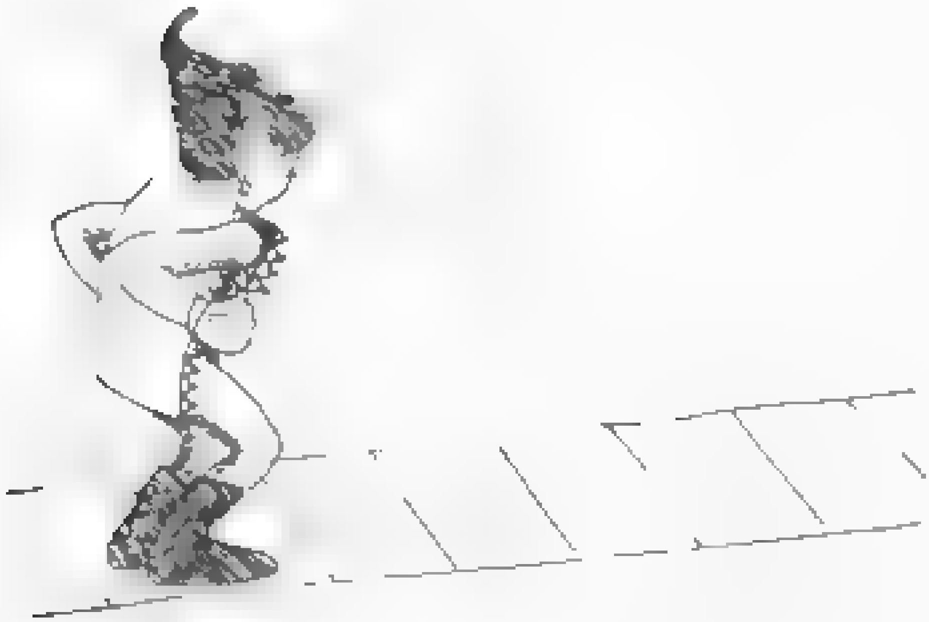
[illegible] 'JUMP', it will jump. Because almost any code can be construed as [illegible] read the byte at the address it finds and presumes that is the

start of the next instruction.

The way the CPU works out the jump instructions is really quite [illegible] tells it where to find the next instruction to be executed. In the [illegible] are to the instruction to the program counter

Thus if it meets a 2-byte instruction, it adds 2, while a 4-byte instruction will make it add 4 to the program counter.

[illegible] contents of the program counter with whatever value you have specified. That is why you cannot allow any errors to creep in



Long Jumps and Short Jump

[illegible]
jump to anywhere the Z80 chip can possibly go.

The disadvantage of the long jump is that:

A. Often we don't want to jump that far but still have to use a 3 byte instruction.

B. We cannot easily relocate the program to another part of memory because we are specifying the absolute address.

[illegible]
can be specified in one byte!

Relative Jump Instruction

[illegible]

JR d

where d is the relative displacement.

[illegible]
example. These conditional jumps are written a

JR cc, d

where cc is the condition to be met.

The value of the displacement 'd' is added to the [illegible]
counter [illegible]

This means it takes the present value of the prog [illegible]
range -128 to +127.

[illegible]
which would be executed if the condition was not met.

[illegible]
after the relative jump!

Eg In a program such as

Location		Code
32000		ADD A,B
32001		JR Z,02H
32003		LD B,0
32005	Next	LD HL,4000H

Load byte at 32000
Because the byte is only a 1-byte instruction so set program counter to 32001.
Execute instruction.
Load byte specified by Program Counter (32001)
Byte is part of 2-byte instruction so add 2 to Program Counter to make it 32003
Get next byte to complete instruction
Execute instruction
cified by Program Counter (32003)
2-byte instruction so add 2 to
(now equal to 32005)
Get next byte to complete instruction
Execute Instruction

At location 32001 the program encounters the Relative Jump

he CPU does nothing.

In general, the CPU executes jump instructions as follows.

If the zero flag is set, add 2 more to the Program Counter (this would make it = 32005
If the zero flag is not set, do nothing
(Program Counter remains = 32003)

In other words, the relative jump allows us to jump over the instruction "LD B,0" in certain cases

for this instruction. It takes less time to do nothing than to calculate the new program counter.

The CPU will therefore execute either the instruction at 32003 or the instruction at 32005 depending on the zero fl

already mentioned

Exercise

```
        JR  -2
```

Machine Language "For ..... Next" Loop.

You are, I am sure, familiar with the BASIC form of the "For . . . Next" loops

loop using the arithmetic functions and the relative jump

```
                            ;Set counter to 1
                            ;Max. of counter + 1
                            ;C = C + 1
                            ;Increment counter
                            ; Is B = A?
        JR   NZ,LOOP        ; If not loop again
```

This will work, but note the following:

We are tying up 2 register

not set any flags on completion.

A much better way would be if we counted down

We know that we have to do the loop 6 tim   so why not set 'B' to 6 and count down?
This will give us

```
        LD B,6          ;set counter
LOOP    INC C           ; C = C + 1
        DEC B           ;Decrease counter
        JR NZ,LOOP      ;Loop is not finished
```

The Z80 chip has a special instruction which combin   the last two lines above.

This instruction is written

DJNZ d

[illegible] This [illegible] and therefore saves one byte on the above coding.

Because of the existence of this special instruction, the B register is usually used as a counting register.

[illegible]

```
        LD B,10H         ; B=16
BIGLOOP PUSH BC          ;Save value of B
        LD B,0           ;Set B 256
LITLOOP
                         ;Whatever calculation

        DJNZ LITLOOP     ;Done 256 times?
        POP BC           ;Get back value of B
        DJNZ             ;Do bigloop 16 times
```

[illegible]

[illegible] register after each instruction in the above program.

Waiting Loops:

There are times in machine language programs when things happen fast it is necessary to just wait a little while. [illegible] printing thousands of characters a second).

[illegible] instruction

```
          LD B, Count
 WAIT     DJNZ  WAIT
```

[illegible] before proceeding again.

[illegible] when you write

```
  WAIT          JR  WAIT
```

[illegible]

Instructions for Call and Return Group

Mnemonic	Bytes	Time Taken	C	Z	PV	S	N	H
			Effect on Flags					
Call address	3	17	-	-	-	-	-	-
Call cc,address	3	10/17	-		-		-	-
RET	1	10	-	-		-		-
RET cc	1	5/11	-	-	-	-	-	-

Note: cc is condition to be met for instruction to be executed. The following are the conditions which can be used:

Flag	Abbreviation	Meaning
Carry	C	Carry Set (=1)
	NC	Carry Clear (=0)
Zero	Z	Zero Set (=1)
	NZ	Zero Clear (=0)
Parity	PE	Parity Even (=1)
	PO	Parity Odd (=0)
Sign	M	Sign Minus (=1)
	P	Sign Pos. (=0)

Flags Effected

[illegible] instructions.

Timing

[illegible] case of the condition not being met.

Use of Subroutines

[illegible]
as it is in ordinary BASIC programs, if not easier.

In fact, remember that using the 'USR' function in your BASIC [illegible]
have a 'RETurn' instruction to finish.

[illegible]

[illegible]
know the address where the subroutine starts.

This can cause a problem if you sto[illegible] the machine language routines in a variable array, because the [illegible]dress of this variable is not necessarily fixed. It also mea[illegible] that machine language programs that use subroutines cannot [illegible]ly be relocated to new positions in memory.

[illegible]
language equivalent of the basic statement:

    IF (condition) then GOSUB (line)

[illegible]
after returning to where you left off."

[illegible]
of four of the flags.

[illegible]
[illegible]
Parity flag (also overflow flag)
Sign flag

Remember that all these flags are set according to the l[illegible] instruction which affected that particular flag.

It is therefore good practice to have 'CALL' or 'RETURN' [illegible]

```
eg.            LD      A, (Number)
               CP      1
               CALL    Z,One
               CP      2
               CALL    Z,Two
```

```
            CP      3
            CALL    Z,Three
```

the value stored in the location 'number', but note that it assumes that the subroutines do not change the value in Register A !!!

A shorter routine is possible if you know that there are only the above three possibilities for the value stored in 'number':

```
        LD          A,(Number)
        CP          2
        CALL        Z,Two       ; A = 2
        CALL        C,One       ; A < 2 => A = 1
        CALL        Three       ; A > 2 => A = 3
```

Flags and the call instructions do not affect any flags.

Similarly the use of the conditional return from a subroutine is

Instructions for Block Compare and Move Group

Mnemonic	Bytes	Time Taken	C	Z	P/V	S	N	H
LDI	2	16	-	-	#	-	0	0
LDD	2	16	-	-	#	-	0	0
LDIR	2	21/16	-	-	0	-	0	0
LDDR	2	21/16	-	-	0	-	0	0
CPI	2	16	-	#	#	#	1	#
CPD	2	16	-	#	#	#	1	#
CPIR	2	21/16	-	#	#	#	1	#
CPDR	2	21/16	-	#	#	#	1	#

(Columns C to H: Effect on Flags)

Flags Notation

\# indicates flag is altered by operation
0 indicates flag is set to 0
1 indicates flag is set to 1
  indicates flag is unaffected

Timing

[illegible]
shorter time indicated is for the case of the instruction terminating – eg. for CPIR, either BC = 0 or A = (HL).

Block Operations

You should [illegible] be very fam[illegible] with the [illegible] [illegible] learning [illegible] [illegible] language you know [illegible]

This [illegible] very useful [illegible] [illegible] with what you already know.

[illegible] language program'

The instructions covered in this chapter are by their very nature [illegible] a block of memory rather than just single 8-bit byte.

Let's start with the simplest of these.

CPI

[illegible] It is in fact an extended compare.

It is read in English as "compare and increase". (You will [illegible] instruction.)

[illegible] next location ready for a repeat.

[illegible]

```
Search      CPI
            JR      NZ, Search
```

[illegible]

Unfortunately this is not such a good idea because un[illegible] is found the program will never end! Fortunately the des[illegible] [illegible] automatically decreases BC'

W[illegible] the length [illegible] the block we wish to

search through and thus specify an end to the search.

[illegible]
is less than 255 bytes long, so that the BC count would only be stored in the C register, we could write

```
Search      CPI
            JR   Z, Found
            INC  C
            DEC  C
            JR   NZ, Search
Notfound    .
            .
Found       .
```

[illegible]
register

[illegible]
instruction

CPD

which is read in English as "compare and decrease". The decrease refers to HL of cour[illegible] and the effect on BC is still the same'

Even more powerful than these two instructions are the real supermen

[illegible]

These are read as "compare, increase and repea[illegible]" and "compare, decrease and repeat".

[illegible]
memory until either a match is found or the end of block is reached. (Naturally we have to specify A, HL and BC before
[illegible]
match found in middle of block or no match at all) we have to ensure we use some code at the end to differentiate between the two possibilities.

[illegible]
language, CPIR and other similar instructions can be very time consuming instruction

second.

that the screen is displayed every 1/50th of a second or so you realise that it can be significant.

The remaining block operations are along the lines of "Move it, Mate".

These are

    LDI   LDIR
    LDD   LDDR

Obviously part of the "load" family these are read as

following set of actions:

    Load (DE) with (HL)
    Increment DE, HL
    Decrement BC

Note that this is the only instruction that will load from one

register first.

clever - this way you never forget which register holds the destination address'

one where the information is and the one where the information is going) overlap

Suppose we are using this instruction in a word processing application, and we want to delete a word from a sentence.

    The big brown dog jumped over the fox.
          7 9 1 3 5 7 9 1 3 5 7 9 1 3 5 7 9

If we want to delete the word 'brown' all we need to do is to move

the rest of the sentence to the left by 6 characters.

DE = destination = character 9
HL = source = character 15
BC = count = 24 characters

Let us start with LDI after one instruction we have

```
original = The big brown dog jumped over the fox.
move one char:     d <---d
```

[illegible]

and HL = 10, DE = 16, BC = 23.

After 2 more instructions
    The big dogwn dog jumped over the fox.

And after all the instructions have been completed:
  The big dog jumped over the fox.e fox.

[illegible]
original sentence and increasing BC to say 30.

[illegible]
overwrite the information we want to shift:

```
eg.     HL = Source        = Character 9
        DE = Destination   = Character 15
        BC = Count         = 24 Characters
```

After one instruction we would have

```
original = The big dog jumped over the fox.e fox.
move char          d---> d
new      = The big dog judped over the fox.e fox.
```

After 6 instructions we would have:
    The big dog judog juver the fox.e fox.

ood. But another three gives
g dog judog jud og the fox e fox.

[illegible]
time yourself by hand.

It is therefore better to use the 'LDI' instruction, with the DE
[illegible]
information woll not be overwritten in the move.

[illegible]
shift thousands of bytes around very quickly.

Exercise

[illegible] memory to the screen.

Note how the 32 first byt[illegible] in the screen are arranged.

Now try 256 bytes, then 2048 bytes.

Instructions that are less frequently used

Register Exchanges

[illegible] locations.

[illegible] arithmetic or counting by themselves.

The first instruction is

```
        EX      AF,AF'
```

[illegible] AF'.

The next general swap gloves instruction is

```
        EXX
```

[illegible] follows:

```
     B    C             B'   C'
     D    E    ( ↔ )    D'   E'
     H    L             H'   L'
```

[illegible] makes it limited in use. This is because it acts on all the [illegible]

[illegible] EXX

[illegible] the lines of

```
            PUSH      HL
            EXX
            POP       HL
```

[illegible] to work with.

The last instruction in this group does not [illegible] swap gloves type

```
            EX      DE,HL
```

[illegible]
of DE.

This instruction is indeed very useful, because as we saw HL is a [illegible]
when the value we want to manipulate is in DE.

Bit, Set and Reset

[illegible] the manipulation of 8-bit or 16 bit numbers.

[illegible] of instructions

[illegible] entire 8 bits of that memory location or register.

[illegible] logical operators

[illegible] any bit "on" or "off" at will, or even just look at a specifi[illegible] bit to see what its status is.

Let us look at the first set of instructions

```
SET n, r
SET n, (HL)
SET n, (IX + d
SET n, (IY + d
```

The "SET" instruction turns "on" ([illegible] = 1) the bit numbered 'n' (using the notation 0 - 7) in regis[illegible]r 'r' or in the specified memory location.

No changes are made to any of the flags.

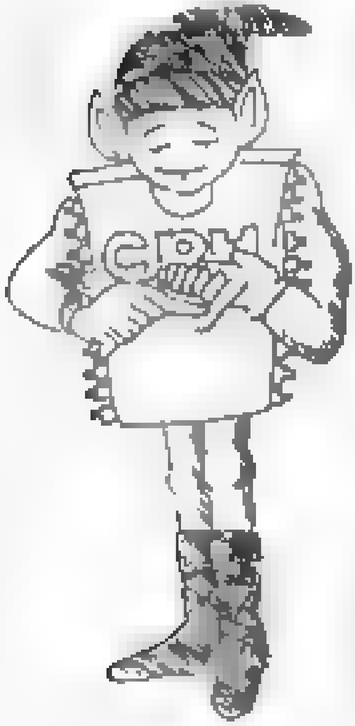
[illegible] it turns the bits "off" (ie. = 0).

[illegible] indicated bit.

[illegible]

```
If Bit   0    then   zero flag is set on ( = 1)
If Bit   1    then   zero flag is set off ( = 0)
```

[illegible]
If the bit is zero, then the zero flag is raised; if the bit is on, then naturally the zero flag would not be raised.



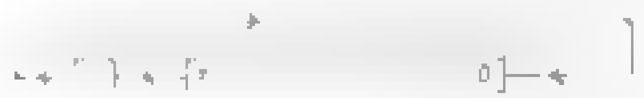
Rotates and Shifts

[illegible] can shift those registers any way you like.

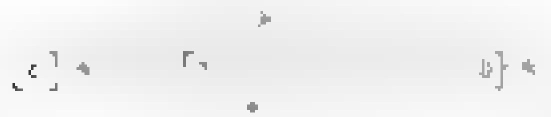
The trick is to differentiate between the various shifts and [illegible] (0 – 7).

[illegible]

[illegible] will be made clear later in this chapter):



[illegible] round'. An example of this is the 'RLCA' instruction:



[illegible]

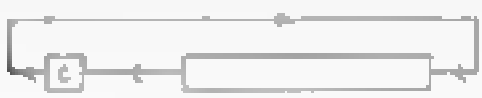
Left Rotations

There are basically two types of left rotations

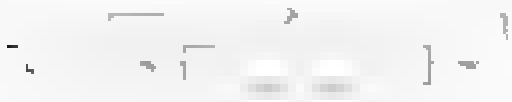
* ROTATE LEFT REGISTERS – this is a 9-bit cycle rotation as illustrated above for 'RLA'

  RLA – "Rotate Left Accumulator"

  RL r – "Rotate Left Register r"



RLCA - Rotate left circular "A"
RLC r Rotate left circular 'r'
RLC (HL) - Rotate left circular (HL)
RLC (IX + d) - Rotate left circular (IX + d)
RLC (IY + d) - Rotate left circular (IY + d)



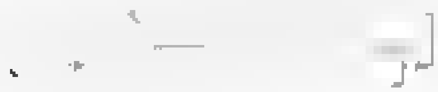
SLA - Shift Left Accumulator

g as nothing is transferred to the accumulator. (Think
if A = 0 H).

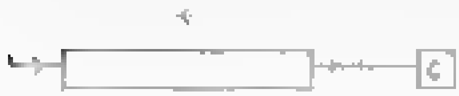
RIGHT ROTATIONS

and rotations can be spinned to the right as to the left.

RRA - Rotate Right Accumulator
RR r - Rotate Right Register



```
RRCA      -    Rotate Right Circular 'A'
RRC r          Rotate Right Circular 'r'
RRC (HL) -     Rotate Right Circular (HL)
RRC (IX+d)-    Rotate Right Circular (IX+d)
RRC (IY+d)-    Rotate Right Circular (IY+d)
```



A similar shift right is available as for shift left:

SRL r - Shift Right Logical Register 'r'

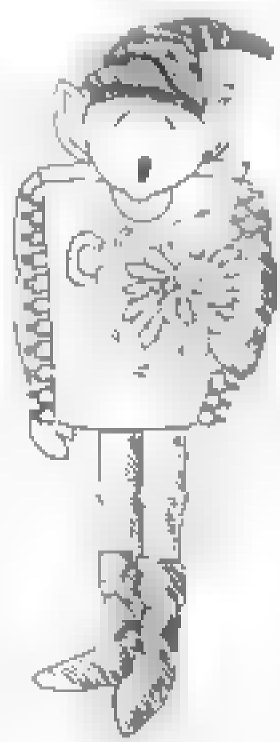
In this case this is pure division by 2 as long as we are using

255).

gative numbers by setting bit 7 to 1 (ie. giving us a range of

SRA r - Shift Right Arithmetic 'r'

As you can see this is also a division by 2 but it preserves the sign bit.



In and Out

[illegible] simple [illegible] machine language programming.

[illegible] needs to get information from [illegible] and [illegible], such as from the keyboard or from the cassette player.

[illegible] The [illegible] is prepared [illegible] know and doesn't care to know how a cassette player works.

[illegible] number [illegible] made by the hardware manufacturer [illegible] there is only the keyboard, [illegible] the cassette player.

[illegible] know [illegible] the data [illegible] or going out, it's an 8-bit byte.

The keyboard and the cassette player [illegible] side of [illegible] to get data in from the keyboard you use the instruction

```
IN A,(FE)
```

[illegible] your [illegible] how the 40 keys of the keyboard are arranged so as to be represented by 8-bit bytes.

[illegible] examined'

The keyboard is divided [illegible] two blocks of 5 keys.

```
        2  3  4  5    6     8  9  .     .
    .  Q  W  E  R  T    ,  I  [     P
       A  S  D  F  G    .  J  K  L N/L (=6
       FT Z  X  C  V    B  N  M  . SPC (=7
```

You can see that there are 8 blocks of letters and we should therefore be able to correlate this with the 8 bits of 'A'.

This is in fact the case

All of the bits of 'A' are set to 'ON' except for one bit which specifies the block to be read.

[illegible] determines which piece of information it gets.

Thus to read the keys in the block "1 2 3 4 [illegible], it is bit 3 of 'A' which should be off:

A = 1 1 1 1 0 1 1 1 = F7

The contents of the keyboard are returned in 'A' with the information coming into the lower bits of 'A'

i.e. Key '1' -) Bit 0 of 'A'
Key '2' -) Bit 1 of 'A'

If block 4 was chosen instead (i.e. A = EFH) then the information would come in as

Key '0' -) Bit 0 of 'A'
Key '9' -) Bit 1 of 'A'

[illegible], so that both '0' and '1' would both go to bit '0' of register 'A'.

[illegible] at all in one instruction (rather than the two instructions which would be [illegible] one block at a time).

[illegible] information at once

eg. A = 1 1 1 0 0 1 1 1 = E7

Note that both bits '3' and '4' are 'OFF'

[illegible] both would set bit 0 of 'A'.

ie. '1' or '0' -) [illegible]
'2' or '9' -) B[illegible]

[illegible] they belong to different blocks in the keyboard.

Note that if you use the instruction

IN r, (C)

selected

cassette input/output doors.

This is still door FE, as mentioned above. The major problem

instruction path.

The OUT instruction is also used to generate sou on the Spectrum and to set the border colour.

ge 160 of the Spectrum manual discusses the BASIC OUT

bit 4 sends a pulse to the internal loudspeaker.

the border colour, load A with the appropriate colour then execute the OUT (FE),A instruction. Note that this

manual).

(EI instruction).

Creating your own sound

16K of RAM

This is done by bringing the WAIT line low.

The effect of this is that any program that requires exact or regular timing is impossible as it is not possible to predict the

such interruptions occur if the program and data the Z80 is accessing is in the ROM or in the upper 32K of memory.

To summarise this in layman's terms, you can produce sounds and

BEEP routine - see the chapter on the Spectrum's features).

To create sound, you need to send a pulse to turn on the

little while later, you need to send another pulse to turn it off. Then a little while later, on again.

In this way sound is created. The total length of time between

you a minimal degree of control over volume.

Note that you must use a value of A for on and off such that the

pattern similar to the LOADing pattern.

Exercise

frequency.

BCD Representation

B[illegible] stands for binary [illegible]ded decimal. This is a way of repr[illegible] information in decimal format.

[illegible] digits from [illegible] four bits are necessary [illegible] this representation.

[illegible] bits are needed to encode a decimal digit [illegible]

eg. 00000000 is BCD representation for decimal 00.
[illegible] 99. What is the BCD representation for "58"? "10"?
Is "10100000" a valid BCD representation?

BCD ARITHMETIC

[illegible] convention in representing numbers can lead to [illegible] problems in addition and subtraction.

[illegible]ding the following

```
BCD    08    0000 1000
BCD    03    0000 0011

BCD    11    0000 1011
```

[illegible] "DAA", called "decimal adjust arithmetic" must [illegible] than 9).

[illegible] illustrated by the same example [illegible] added to the second BCD digit.

The "half carry flag", H is used to detect this carry.

```
LD  A,  12H      ;load literal B[illegible]
ADD A,  24H      ;add literal BCD "24"
DAA              ;decimal adjust result
LD  (addr), A    ;store result
```

[illegible] BCD representation in your programming [illegible] to know that the Z80 chip [illegible] supports this representation and the DAA instruction will make the [illegible] of a small group of BCD users simpler.

Interrupts

An [illegible] sent to the microprocessor, which may [illegible] at any time and will generally suspend the execution of the current program (without the program knowing it).

Three interrupt mechanisms are provided in the Z80. [illegible] the non-maskable interrupt (NMI) and the [illegible] interrupt (INT).

From a programming point of view we will only look at the usual maskable interrupt (INT).

The DI (disable interrupt) instruction is used to reset (mask), while the EI (enable interrupt) instruction is used to set (unmask).

[illegible] interrupt [illegible] the [illegible] program [illegible] RET [illegible] return [illegible] interrupt.

[illegible] interrupt [illegible] and [illegible] This interrupt allows the keyboard to [illegible]

[illegible] programs [illegible] your own routine to do so.

[illegible]

Restarts

This is rather a [illegible] from the 8080 implemented for compatibility. That is why you will be unlikely to use RST instructions in your program.

The RST performs the same actions as a CALL, but allows a jump to only one of eight addresses in the first [illegible] memory locations: 00H, 08H, 10H, 18H, 20H, 28H, 30H or 38H.

The advantage of the RST instruction is that it [illegible] only one byte. The RST instruction also takes less time than a CALL instruction.

The disadvantage of RST instruction is that it can only be used to call one of the above eight possible locations.

As all these locations are within the ROM, [illegible] own programs [illegible] ROM [illegible] if you know [illegible] RST instructions

[illegible]

Programming Your Spectrum

Planning Your Program

[illegible] allows you to do anything at all.

[illegible] Fortran or Cobol or any other language can be done in machine languag[illegible]

[illegible] be the faster on[illegible]

This total flexibility can however also be a tr[illegible], to the unwary [illegible] are no checks on whether the statement is a legal one.

[illegible] or another, the Z80 chip will process everythin[illegible]

[illegible] totally illegal in any higher level language.

[illegible]

[illegible] without doing any coding for a long time.

Suppose you wanted to write a lunar lander program [illegible]

```
INSTR          Display instructions
               Jump back to INSTR till ENTER pressed
[illegible]    [illegible]
LAND           Move Lander
                    If fuel finished go to CRASH
                    Jump back to LAND if not ground
[illegible]         [illegible]
                    Jump back to INSTR for next GO
[illegible]    [illegible]
                    Jump back to INSTR for next GO
```

Notice how this 'program' is written totally in English. At this stage, no decision has been made whether the program is to be

that decision - the concept of the Lunar Lander program is not dependent on the coding

Now comes the part of logic testing.

you wish to see included in the program are covere

things be put into subroutine

way to finish the program

arcade machine, but in your program you may decide you would like to be able to turn the program off

We now change the last part of the program as follows.

```
GROUND        Print Congratulations
                   Jump to Finish

FINISH        Ask player if finished
              If not, jump to INSTR
              If yes, STOP
```

Note that we have used labels to describe certain lines in program. These are very valuable, the more so if you choo labels which are descriptive in their meaning.

thing to one of the lines or modules above.
This is why this approach is called the top down approach.

For example we can expand the 'finish' module above:

```
FINISH            Clear screen
                  Print "Would you like to stop now?"
                  Scan keyboard for input
                  If input = yes then stop
                  Jump to INSTR
```

run a particular module on its own, so that it is ready for the final program.

Let us go down one level further again, and look at the
                   Clear screen

...ine in more detail.

If you were writing in BASIC, all you would have to say is

900 CLS

but in machine lang e that simple sentence, 'Clear screen' can be deceptive.)

We might therefore do something like

Fill next 6144 positions with blanks

s clear screen routine is meant to do and what it will actually

and so on.

routine above will clearly be inadequate.

to work on the attribute file well. (Note how much more complex certain t ks can be in machine language than in BASIC.)

efore need to expand the program to read

Find screen beginning
next 6144 bytes with b nks
ttribute file beginning
next 768 bytes with pa r/ink desired

The next level down is the one where you must finally do the coding, so let us look at filling the screen with blanks

```
CLEAR       LD   HL,SCREEN      ;Screen start
            LD   BC,6144        ;Bytes to clear
            LD   D,0            ;D=blank
LOOP        LD   (HL),D         ;Fill blank
            INC HL              ;Next position
            DEC BC              ;Reduce count
            LD  A,B
            OR C                ;Test if BC = 0
            JR NZ,LOOP          ;Again if not end
```

this way build up very complex programs indeed.
By the way, you no doubt understand now why machine language

language programs?

There are more ways than one to write any particular routines. Let us look at the simple clear screen routine written above.

This could be handled by several different approaches.

Exercise 1

only so that we may make use of the 'DJNZ' instruction?

Exercise 2

blanked using the more powerful 'LDIR' instruction?

Think carefully of what 'LDIR' does: it is not always neces ry to have 6144 blank positions elsewhere

Answers:

More than one possible answer can be   ght" - the only test is   it work? In other words does it do what YOU want?

Using DJNZ

```
CLEAR      LD   HL,SCREEN
           LD   A,0
           LD   B,24          ;Set B=24
BIGLOOP    PUSH BC            ;save value
           LD   B,A           ;Set B 256
LITLOOP    LD   (HL),A        ;
           INC  HL            ;Fill in 256 blanks
           DJNZ LITLOOP
           POP  BC            ;Get back value of B
           DJNZ BIGLOOP       ;Do it until end
```

We have been able to use 24 times 256 (=6144) to clear the screen

Points of note are
We can set B = 0 to go through the DJNZ loop 256 times. (Why?)
This procedure would not normally be used in a program unless we were also using register C for other purposes.

Using LDIR

```
CLEAR       LD HL,SCREEN      ;Source
            PUSH HL
            POP DE
            INC DE
            LD BC,6144
            LD (HL),0
            LDIR              ;Move it
```

Note that we have found DE = HL +1 by getting DE = HL and increasing DE. This can be achieved more easily by loading the value of SCREEN +1 into DE directly but this requires 1 more byte!

The reason this LDIR works is because we are using the fact that the data is overwriting the block to be written as we proceed. This

Move Chapter.

If you add up the memory required, the first method requires 14 bytes, the second 16 bytes, and the last 13 bytes.

Features of the Spectrum

[illegible] machine language programs [illegible] it.

Input — keyboard

As far as input to the Spectrum is concerned, we will ignore cassette input and concentrate on the keyboard.

The keyboard is the only input which provides real-time [illegible]

[illegible] eight rows and five columns as in appendix A.

[illegible] In their normal state (when they are not pr[illegible] n a high mood ie. the inter[illegible]

When a [illegible]ticular key is pre[illegible] [illegible]nding to that key will be re[illegible]

[illegible] which can be used in machine language programming.

[illegible] (p 160) of the Spectrum manual.

[illegible] listed in the leftmost column of the table in appendix A.

eg. For the "H — ENTER" half-row we load A with value BFH

```
LD  A, BFH
```

[illegible] the INPUT instruction is issued.
eg. The port used is the FFH port

```
IN  A, (FEH)
```

Since there are five keys per half row, we are only interested in the five low order bits of the returned byte in A.

If no key is pressed in that half row, the value of the low order five bits will be (2**4 + 2**3 + 2**2 + 2**1 + 2**0 ie. 16 + 8 + 4 + 2 + 1 = 31).

register A = xxx11111 when no key is pressed.

If we want to test whether the rightmost key is pressed, we have to see whether that bit is low.

There are two ways to test tha

i. Use Bit test instructions, eg BIT 0, A
   If the bit is low (not set) then the Zero flag will be set.
ii. Use Logical AND instructions AND 1
   If the bit is low (not set) then the result will be zero and the Zero flag will be set.

Th[illegible] method is eas[illegible] to be [illegible] especially when we want to test a spe[illegible] directly [illegible] Bit [illegible] test [illegible] that half row we [illegible] two Bit test instructions and possibly two relative jumps.

eg To test bit 0 and bit 1 using the first method

```
        BIT 0,  A       ;test bit 0 of A set or not
        JR  Z,  NPRESS  ;jump if not pressed
        BIT 1,  A       ;test bit 1 of A set or not
        JR  Z,  NPRESS  ;jump if not pressed

        do whatever if both are pressed

NPRESS
```

The second method of testing using logical AND requires a little more [illegible]. To test bit 0 we use "AND 1"; to test bit 1 we use "AND 2"; to test bit 2 we use "AND 4" and so on.

To test two keys, we use "AND x" where x is the sum of the values we will use when testing each one key individually.

eg. To test both bit 0 and bit 1 of A are set

```
        AND     3       ;test both bit 0 and bit 1
                        ;is set
        CP      3       ;test if both set
```

```
        JR      NZ,NBOTH    ;jump if not both pressed
```

To test if either bit 0 or bit 1 of A are set

```
        AND     3           ;test either bit 0 and bit
                            ;1 is set
        JR      Z,NOTONE    ;jump if not one is pressed
```

Ex

[illegible]

for your Spectrum.

You will need to

a. [illegible]heck the row address that needs to be loaded into A.
b. [illegible]nd it to the input port FEH.
c. Test the bit that is set by the (ENTER) key.

Output   Video Screen display

The Video screen display is the main source of output for the computer to communicate to the user.

screen memory of the Spectrum is organised

```
210040   LD  HL,4000H        ;load HL with start of
                             ;display file
36FF     LD  (HL), FFH       ;fill that screen location
110140   LD  DE,4001H        ;load DE with next byte
                             ;in display
010100   LD  BC,1            ;BC contains number of
                             ;bytes to be transferred
EDB0     LDIR                ;move a block length BC
                             ;from (HL) to (DE)
C9       RET                 ;end of progr
```

transferred from (HL) to (DE).

Now change the fourth line to read LD BC, 31 (011F00). You may be

Note how a very thin line has been drawn across the top of the screen. The first 32 bytes of the screen memory relate to the first byte of each of the first 32 characters.

surprised. The next byte after the 32nd one is not on the second row of dots on the screen. It is the first byte of the 32nd character! And so on up to the 256th character.

that line to LD BC, 2047 (Ø FFØ7) and run the program. You will find that the top third of the screen only has been filled.

You can experiment with this, using different values for BC up to LD BC, 6143 (Ø,FF17). In this way you can watch the way Spectrum organises the screen.

The screen memory is actually divided into three lots.

i. Memory 4000H - 47FFH (= =) first eight line
ii. Memory 4800H - 4FFFH (==) second eight lines.
iii. Memory 5000H - 57FFH (===) third eight lines.

Not only that, but you will recall that each character of the Spectrum is composed of eight 8-bit bytes which makes up 64 dot

eg. For the character " ! ", its character representation is

0	00000000	0H
16	00010000	10H
16	00010000	10H
16	00010000	10H
6	00010000	10H
0	00000000	0H
6	00010000	10H
0	00000000	0H

the first 256 bytes from 4000H to 40FFH correspond to the first byte of each of the 256 8-byte character of the first eight lines.

Then the next 256 bytes from memory location 4100H to 41FFH correspond to the second byte of each of the 256 8-byte character of the first eight lines and so on

Thus

First character of the screen is

1st byte	4000H
2nd byte	4100H
3rd byte	4200H
4th	
5th byte	4400H
6th byte	4500H
7th byte	4600
8th byte	4700H

[illegible] isn't it? But we have to accept the Spectrum the way it is [illegible] it.

Can you write down the eight bytes that correspond to the 31st character of the third line of the screen? You can refer to Appendix B, the screen memory map.
405EH, 415EH, 425EH.....475EH..

To follow on the concept we have developed about the screen display, the memory locations that correspond to the first character of the second eight lines lot is
4800H, 4900H, 4A00H, 4B00H, 4C00H, 4D00H, 4E00H, 4F00H.

Similarly, the first character of the third eight lines lot has its eight bytes in memory locations
5000H, 5100H, 5200H, 5300H, 5400H, 5500H, 5600H, 5700H.

There are some advantages, however, in using machine language. The apparent complexities are worth overcoming. As a trivial example, in BASIC, if you try to PRINT into the input section of the screen (the bottom two lines), the BASIC system will object most violently. But in machine language you have full access to the

whole screen.

If you observe the [illegible] display organisation more closely, you will see that the High Order Byte of First Byte (HOBFB) of each character determines which lot of the three memory portions the character is in.

For example, if 40H =(HOBFB (41H   char is in first
                                        eight lines lot
             if 48H =(HOBFB (49H   char is in second
                                        eight lines lot
             if 50H =(HOBFB (51H   char is in third
                                        eight lines lot

[illegible] the lower [illegible] bits of the HOBFB High Order [illegible] determines which [illegible] eight bytes of the character it belongs.

The [illegible] Appendix 8 [illegible] memory [illegible] and the display screen (if any!!*).

Try the following example.:
Suppose we are given an addr[illegible] as 4A36H. The High Order Byte of the address is 4AH so,

i. we know that it is within the screen display memory since its value is in between 40H and 58H.

ii. Its binary representation is 01001010

iii. from the lower three bits we know that it belongs to the third byte of a character position on the screen.

iv. If we made the lower three bits zero, then the value [illegible] 48H. Thus we know this belongs to the second eight lines lot, ie the middle portion of the screen display.

The [illegible] we can reach is that the byte given refers to the third byte [illegible] the middle portion of the display memory.

Which character of the middle portion does the byte belongs to? To answer this question, we will need to know the value of the Low Order Byte of the address.

We know that LOB of the address is 36H. So the address refers to [illegible] 54th position away from the first character of the middle portion.

S[illegible] has [illegible] characters, the posit[illegible]

the second line of the middle screen di[illegible]ay portion and is the (54-32+1)th character of that line.

[illegible]

of the 23rd character of the 10th line from the start of the [illegible]

Exercise

Which byte of which character do[illegible] the address 364FH refer to?

Output Video display attribute

The display attribute memory is easier to understand than the display memory because it has a one-to-one relationship with the screen display characters.

The attribute file is located in memory from 5800H to 5AFFH. It is 768 bytes, which correspond to 24 lines of 32 character each. In other words, there is one attribute byte for each character position.

Thus, 5800H corresponds to the attribute of the first character of the first line, 5801H the second character, 5802H the third,....581FH the thirty second character of the first line.

Similarly, 5820H holds the attribute of the first ch ter of the second line, 5840H of the third line, ... and 5AE0H the of the first character of the last line of the screen.

We know that for each character position on the screen, there is a corresponding attribute byte in the attribute memory, made up as follows

```
        attribute by e        b b bbb bbb

    bit 0 - 2        represents the ink colour of the
                     character 0 to 7.
    bit 3 - 5        represents the paper colour of the
                     chara ter 0 to 7.
                          t if 1, normal if 0.
                                         h if 0
```

What is the address of the attribute byte that c
first byte of the middle screen section? What is the
the first byte of the third section? Answers are given on the next page, but try to work it out for yourself.

xercise

Can you write a subroutine that converts a given address on the screen to its corresponding attribute address
e.g. 4529H

You must in effect determine which character of this screen th s belongs to, and then add this to 5800H

The following program shows a short method of achieving this:

```
 L  HL, 4529H          ;oad the given address to HL
LD  A                  ;oad the high order byte to A
                       ;trap bits 3 and 4 to
                       ;determine which portion of the
                       ; creen the address belon
                       ;h.ft right accumulator
                       ;three times - ie divide by 8
                       ;result can either be 0,1 or 2
                       ;depending whether H was
                           48H, 50H or 50H
                       ;transform to attribute memory
                       ; contain attribute address
                               58H, 59H or 60H
                            ..ins the same
```

You may he   to think about this for a while!

The way the pro   m works is related to the answer of the first exercise

```
1st char. of 1st screen section = 4000H   A                       58 .
1st char. of 2nd screen section = 4800H   A                       5900H
 st char. of 3rd screen section = 5000H   A                       5A00H


2nd char. of 1st screen section = 4801H   Attribute add           5901H
                etc.
                         etc.
```

This should make things a little clearer!

Output - Sound

offers is sound. It would be a waste if we didn't make full use of this facility.

generating sound.

1. Sending signals to the cassette output port 254 for certain duration of time using the OUT instruction 254.
   eg OUT (254), A
ii. Set HL, DE to certain values and call the ROM sound routine used to generate sound.
   The input parameters are
   DE - duration in sec * frequency
   HL - (437,500 / frequency) - 30.125
   Then
   CALL 03B5H.

The first way of sound generation has the advant of being free from any ROM call. It is shorter in terms of time to execute. But there is

ave a 48K machine.

using this method, but it will not be 'clean sound'. You have to use the second method of sound generation (of calling the ROM routine) to get that result.

your Spectrum user manual.

program. You can think of the DE register pair as holding a value for the duration of the sound, and HL a value for the frequency.

sound you want.

The limitation of this method of course is that you are restricted to whatever sounds you can create with the BEEP command.

Monitor Programs

EZ-Code Machine Language Editor

This is a machine code monitor program that allows you to

i. INPUT your machine language program module in
   either a fully assembled format
   or a semi-assembled format with all relative jump and absolute jumps expressed in the form of line number.

ii. LIST the source input program module.

iii. DUMP the input program module into the specified memory address.

iv. EXAMINE a range of memory locations.

v. SAVE EITHER the "source modul
   OR the dumped program in fully machine code format

vi. LOAD a saved "source program" from the ca

vii. RUN the dumped chine program module

not need to calculate relative or absolute jumps

200 instructions

to wipe off the EZ-code program.)

CONCEPT behind the EZ-code

of a BASIC program.

code) has a line number and up to 4 bytes of machine code.

A major benefit is therefore the ability to "edit" any line. The "source program" can also be SAVED separately to tape, allowing work in progress to be saved.

A major innovation in this program is the ability to insert relative jumps or absolute jumps without having to calculate the

number you wish to jump to'

the scope of a relative jump.

to memory.

EZ-code Instruction Summary

Note that the first question the program will ask you is

"Loading address".

This cannot be below 31500.

**** Entering LINEs ****

i. To ENTER lines of "source program

(line no) blank)(maximum of 4 bytes in Hexadecimal)
ENTER)

eg. 1 210040 will insert the machine code instruction
LD HL,4000H into line number 1.

ii. To EDIT a lin

(line no)(blank)(retype new bytes)(ENTER

eg. 1 210140 will change line number 1 to the instruction
LD HL,4001H.

iii. To DELETE an instruction line

line-no)(ENTER

eg. 1 (ENTER) will delete line number 1.

iv. To specify RELATIVE or ABSOLUTE jump

(line-no)(blank)(jump instruction)("lower case "L")
(line no (ENTER

eg. 1 c312 represents the instruction JP to line 2.
2 18.1 represents the instruction JR to line

**** COMMANDS ****

i. dump ENTER
* dump the source listing into the memory starting from the specified LOADING addre
* this must be done before running the machine code program.

abbreviation: du

ii. exit ENTER
* exit from the EZ-code and re-enter BASIC system.

abbreviation: ex

iii. list(ENTER
* list the first twenty-two instruction lines of the source listing
* press any key except "m" and "BREAK" to continue listing

abbreviation. li

list# ENTER
* list twenty-two lines of the source listing starting from line number #, a number between and 2 0 inclusively.

abbreviation NO ABBREV

iv. load ENTER
* load a source listing module from the cassette replacing the existing module.

abbreviation: lo

v. mem ENTER
prompt: Starting address:
* enter memory address you want to start displayin from.
* can be from 0 to 32767 for 16K Spectrum or 0 to 65535 for 48K Spectrum
* press "m" to exit memory examine mode.

abbreviation: me

vi. new(ENTER
* clear the current module and re-run the EZ-code.
* this is useful when you want to start coding in another program module
abbreviation ne

vii.run(ENTER)
* run the dumped program module from LOADING address you specified when you start running the EZ code program or when you LOAD a new source listing
abbreviation. ru

* save either the source listing or dumped machine code onto cassette.
prompt. Enter name
enter the name you want to us
Source or Machine code (s or m)
enter s for source listing saving
enter m for machine code saving
Start tape, then press any key.
make sure that the cassette 1 is properly

press any key when the c ette is ready
abbreviation sa

NOTE

1. If you don't want the result of BC register returned after running, change line 3090 to :

restart the EZ code :
Either use RUN and resulting with all varables reinitialise
Or use GOTO 2020 which returns the prompt "Command or Line(###) ".

3. All numeric entry except machine instruction code has to be in decimal format.

4. To enable you to insert additional lines in the current listing, it is good to space out the listing. ie.instead of entering instruction lines as 1, 2, 3 enter as 1, 5, 10 etc
This will makes the input of the module more flexible.

[illegible] on EZ-code

Enter the following code

```
        210040          LD   HL,4000H          ;fill screen
        110140          LD   DE,4001H
        01FF17          LD   BC,6143
        3EFF            LD   A, 0FFH
        77              LD   (HL), A
        EDB0            LDIR
        3E7F    LOOP:   LD   A, 7FH            ;trap BREAK key
        DBFE            IN   A, (0FEH)
        E601            AND  1
        20F8            JR   NZ, LOOP
        C9              RET
```

To enter the above code using EZ-code

```
GEN
        address         [illegible] ENTER
           line#        1 210040 ENTER
        or line ###     5 110140 ENTER
           line ###     10 01FF17 ENTER
                        15 3eff ENTER
                        20 77 ENTER
           line###      25 edb0 ENTER
        or line(###)    30 3e7f ENTER
           line###      35 dbfe ENTER
           line###      40 e601 ENTER
command or line ###     45 2[illegible] ENTER
(This is 2[illegible] then lower c[illegible] "[illegible]", then 30. In other words
        [illegible] NZ, line 30 )
command or line ###     [illegible]0 c9 ENTER
command or              [illegible]st ENTER
command or line ###     dump ENTER
command or line ###, [illegible] ENTER
Starting address  [illegible] ENTER
                        this is the key to exit the memory

command or line#
```

Note how there must be a space after the line numbers.

6ZCODE


```
 100 REM machine
 110 REM machine_code_monitor
 120 GO TO 9000
 130 DEF FN d(s$) = (s$ > "9")*( CODE s$-55)
        +(s$ <= "9")*( CODE s$-48) (s$ > "F")*32
 140 DEF FN o(O$) = ((O$ = "ca")+(O$ = "da")
        +(O$ = "ea")+(O$ = "fa")+(O$ = "c2")
        +(O$ = "d2")+(O$ = "e2")+(O$ = "f2")
        +(O$ = "c3")) ((O$ = "38")+(O$ = "30")
        +(O$ = "28")+(O$ = "20")+(O$ = "18")
        +(O$ = "10"))
1000 REM
1010 REM  ___ LINE_PRINTING_routine ___
1020 [illegible]
1030 LET F = ze : PRINT AT z  ze;
1040 FOR J = pl1 TO pl2
1050 IF C$(J, on) = "__" THEN GO TO 1110
1060 PRINT TAB tr- LEN STR$ J; J; TAB fr; "_";
1070 IF C$(J, tw, on TO on) = "1"
        THEN PRINT C$(J, on)+"_"+C$(J, tw)+C$(J, tr)
     : GO TO 1090
1080 PRINT C$(J, on); "_"; C$(J, tw); "_"
        ; C$(J, tr); "_"; C$(J, fr)
1090 LET F = F+on
1100 IF F = 22 THEN GO TO 112
1110 NEXT J
1120 PRINT AT  , 25; "_______"
1130 RETURN
2000 REM
2010 REM  ___ main_routine ___
2020 INPUT "Command_or_Line(###) :_"; A$
2030 IF A$( TO fr) = "____" THEN GO TO mr
2040 IF A$(on) > "9" THEN GO TO 3000
2050 LET k$ = "" : FOR K = on TO fr
2060 IF A$(K TO K) = "_" THEN GO TO 2090
2070 LET k$ = k$+A$(K TO K)
2080 NEXT K
2090 IF K = 5 OR VAL k$ = ze OR VAL k$ > ln
        THEN GO TO mr
2100 LET J = VAL k$ : LET n = J
     : REM line_number_must_be_3_bytes
2110 LET A$ = A$(K+on TO )
2120 LET k$ = ""
2130 FOR K = on TO LEN A$
2140 IF A$(K TO K) <> "_"
        THEN LET k$ = k$+A$(K TO K)
2150 NEXT K
2160 LET A$ = k$
2162 IF A$(on) =      THEN GO TO mr
```

```
CLS : FOR I = on TO 7 STEP tw
LET k = INT (I/tw+on)
LET C$(J, k) = A$(I TO I+on)
NEXT I
IF C$(n, on) = "  " THEN GO TO 2250
IF n < TP THEN LET TP = n
IF n > BP THEN LET BP = n
GO TO 2?2?
IF n <> BP THEN GO TO 2?8?
IF BP = on OR C$(BP, on) <> "  "
    THEN GO TO 2?2?
LET BP = BP-on : GO TO 22??
IF n <> TP THEN GO TO 232?
IF C$(TP, on) <> "  " THEN GO TO 232?
IF TP <> BP AND TP <> ln THEN LET TP = TP+on
: GO TO 22??
LET TP = on
LET pp = n
IF n < TP THEN LET pp = TP : GO TO 238?
LET numlp = ze
IF pp = TP OR numlp = 1l THEN GO TO 238?
IF C$(pp, on) <> "  "
    THEN LET numlp = numlp+on
LET pp = pp-on : GO TO 2350
LET pl1 = pp : LET pl2 = BP
GO SUB 1000
: REM print a block of lines
GO TO mr
REM
REM  [illegible] Commands**************** IBL
LET k$ = A$( TO tw
    k$ = "d " THEN GO TO 5000
    k$ = "e " THEN STOP
    k$ = "li" THEN GO TO 4000
    k$ = "lo" THEN GO TO 7000
    [illegible]    THEN GO TO 6000
    [illegible]    THEN RUN
    [illegible]    THEN PRINT USR R
    [illegible]    THEN GO TO 8000
[illegible]
REM
REM  [illegible] routine*********** IBL
LET pl1 = TP : LET pl2 = BP
LET n1 = CODE A$(6 TO 6)
IF LEN A$ > fr AND n1 > 47 AND n1 < 58
    THEN LET pl1 = VAL A$(5 TO 8)
GO SUB 1000
GO TO mr
REM
REM  [illegible] DUMP routine*********** IBL
CLS : PRINT AT ze, 25; INK on; INVERSE on
    ; FLASH on; "DUMPING" : LET G = R
```

```
PRINT AT on, ze;
FOR J = TP TO BF
IF C$(J, on) = "  " THEN GO TO 5470
IF C$(J, tw, on TO on) <> ":" THEN GO TO 5780
POKE G, ze : POKE G+on, ze : POKE G+tw, ze
: POKE G+tr, ze
LET jl = VAL (C$(J, tw, tw TO tw)+C$(J, tr))
PRINT TAB tr- LEN STR$ J; INVERSE on; J
    ; TAB fr; INVERSE ze; " "
    ; C$(J, on)+" "+C$(J, tw)+C$(J, tr)
    ; " =  > ";
IF jl < ze OR jl > ln THEN GO TO 5460
LET CJ = FN O(C$(J, on))
PRINT TAB 17- LEN STR$ jl; INVERSE on; jl
    ; TAB 18; INVERSE ze; " "; C$(jl, on)
    ; " "; C$(jl, tw); " "; C$(jl, tr); " "
    ; C$(jl, fr);
IF ABS CJ <> on THEN GO TO 5460
LET dd = (jl > J)-(jl < J)
LET ja = G : LET dp = ze
IF jl = J THEN GO TO 5?70
LET cl = J+dd
LET nl = ze : IF C$(cl, on) = "  "
    THEN GO TO 5?2?
IF C$(cl, tw, on TO on) <> ":"
    THEN LET nl = on+(C$(cl, tw) <> "  ")
           +(C$(cl, tr) <> "  ")
           + C$(cl, fr) <> "  ")
: GO TO 5 ?
LET TJ = FN o(C$(cl, on))
LET nl = (TJ = on)*(r+(TJ = -on)*tw
IF cl = jl AND dd > ze THEN GO TO 5?7
LET dp = dp+nl
IF cl = jl THEN GO TO 5 70
LET cl = cl+dd
GO TO 5 ?0
IF CJ = on THEN LET ja = ja+dd*dp+(dd > ze)*tr
: GO TO 5 ?0
IF dd > ze THEN LET dp = dp+2
IF dp > 126 AND dd < ze THEN GO TO 5460
IF dp > 129 AND dd > ze THEN GO TO 5460
LET V = 16* FN d(C$(J, on, on TO on))
+ FN d(C$(J, on, tw TO tw))
POKE G, V : LET G = G+on
IF CJ = on THEN POKE G, ja- INT (ja/qk)*qk
: LET G = G+on : POKE G, INT (ja/qk)
: LET G = G+on : GO TO 5?60
IF dd < ze THEN LET dp = -dp
LET dp = dp-tw : POKE G, dp : LET G = G+on
PRINT "ok"
GO TO 5470
FOR I = on TO 7 STEP tw
```

```
5[illegible]  LET K = INT (I/tw+on)
5[illegible]  LET V = 16* FN d(C$(J, K, on TO on))
                + FN d(C$(J, K, tw TO tw))
54[illegible] IF V < ze THEN GO TO 544[illegible]
54[illegible] POKE G, V
54[illegible] LET G = G+on
544[illegible] NEXT I
545[illegible] GO TO 547[illegible]
546[illegible] PRINT "**"
547[illegible] NEXT J
54[illegible] PRINT AT ze, 25; "[illegible]"
              : GO TO mr
6[illegible]  REM
6[illegible]  REM  INV Memory display********** IBW
6[illegible]  INPUT "Starting address ? "; dm
6[illegible]  CLS : PRINT AT ze, ze;
6[illegible]4 LET G = dm : LET F = ze
6[illegible]  LET F = F+on
              : PRINT TAB 5- LEN STR$ G; G ; TAB [illegible]
6[illegible]  FOR I = on TO fr
6[illegible]  LET V = PEEK G
6[illegible]  LET H = INT (V/16)
6[illegible]  LET L = V-16*H
61[illegible] PRINT D$(H+on); D$(L+on); " ";
6[illegible]  LET G = G+on
6[illegible]  NEXT I
61[illegible] PRINT " "
6[illegible]4 IF F <> 22 THEN GO TO 605
6[illegible]  LET k$ = INKEY$ : IF k$ = "" THEN GO TO 615[illegible]
6[illegible]  IF k$ [illegible] AND k$ [illegible] "M" THEN [illegible]
              : POKE 23692, q+-on : GO TO 6050
6[illegible]  POKE 23692, on : PAUSE 20 : GO TO mr
              REM
7[illegible]  REM  INV LOAD******************** IBW
7[illegible]  [illegible]
7[illegible]  INPUT
                  "Load array (Press any key when   ready) "
                  ; k$
7[illegible]4 PRINT AT ze, 25; INVERSE on; FLASH on; "LOADING"
7[illegible]  LOAD "source" DATA C$()
7[illegible]  FOR I = on TO ln
7[illegible]  LET TP = I
7[illegible]  IF C$(I, on) <> "  " THEN GO TO 7100
7[illegible]  NEXT I
7[illegible]  FOR I = ln TO on STEP -1
71[illegible] LET BP = I
71[illegible] IF C$(I, on) <> "  " THEN GO TO 7140
71[illegible] NEXT I
7[illegible]4 PRINT AT ze, 25; "       "
7[illegible]  GO TO 9150
8[illegible]  REM
8[illegible]  REM  INV SAVE******************** IBW
```

```
8020 INPUT "Enter▲name :▲"; n$
8030 IF n$ = "" THEN GO TO 8020
8040 INPUT
         "Source▲or▲Machine▲code :▲(s▲or▲m)"
         ; k$
8050 IF k$ <> "s" AND k$ <> "m" THEN GO TO 8040
8060 IF k$ = "s" THEN SAVE n$ DATA C$() : GO TO mr
8070 INPUT "Starting▲address :▲"; ss
8080 INPUT "Finishing▲address :▲"; sf
8090 LET sb = sf-ss+on
8100 SAVE n$ CODE ss, sb
8110 GO TO mr
9000 REM
9010 REM initialisation
9020 LET ze = PI - PI : LET on = PI / PI
      : LET tw = on+on : LET tr = on+tw
      : LET fr = tw+tw : LET qk = 256
      : LET mr = 2020 : LET ln = 200
9025 BORDER 7 : PAPER 7 : INK on : INVERSE ze
      : OVER ze : FLASH ze : BRIGHT ze
      : BEEP .25, 24 : BEEP .25, 12
9030 DIM A$(15) : DIM D$(tw)
9040 LET TP = ln : LET BP = on
      : REM line▲number▲buffer
9050 DIM C$(ln, fr, tw) : REM holds▲code
9060 PRINT AT ze, 20; INVERSE on; FLASH on
         ; "INITIALISING"
9070 FOR I = on TO ln
9080 FOR J = on TO fr
9090 LET C$(I, J) = "▲"
9100 NEXT J
9110 BEEP .01, 20
9120 NEXT I
9130 PRINT AT ze, 20; "▲▲▲▲▲▲▲▲▲▲▲▲▲▲"
9140 LET D$ = "0123456789ABCDEF"
9150 CLS : PRINT "Lowest▲address :▲"; 31500
9160 INPUT "Loading▲address :▲", R : PAUSE 1
9170▲ IF R < 31500 THEN GO TO 9160
9180 CLS : GO TO mr
```

Hexload Machine Code Monitor

[illegible] program [illegible] the memory [illegible] the memory onto cassette and [illegible] cassette to memory.

[illegible] Hexload [illegible] less than 200 instructions.

[illegible] save each module as machine code on cassette.
[illegible] load [illegible] their appropriate memory locations.

We will actually apply this technique as we develop the FREEWAY FROG program.

Concept behind Hexload

The concept behind Hexload is extremely simple.
[illegible] RAMT [illegible] 26999

[illegible] for 48K Spectrum.

[illegible]
It offers basic monitoring functions like.

    WRITE   onto memory in Hex format
    SAVE   from memory to cassette
    LOAD   from cassette to memory
    LIST   memory contents from a starting address
    MOVE   memory contents from one locations to another.

Hexload Instructions Summary

1   WRITE

        Write code in HEX format onto the memory.

Procedure

        Input start of memory where you want to write to in decimal format in response to the prompt.

The address is limited to 27000 – 32558 for 16K
27000 – 65346 for 48K

eg. Write to address 27000(ENTER)

b. Enter codes in hex format.

c. Press "m" to return to main menu.

2. SAVE

Save memory to cassette.

Procedure.

a. Input memory from which saving starts, can be any address 0 – 32767 for 16K
0 – 65535 for 48K

Input number of bytes to be

Input name of the module to be saved

[illegible]

Option of verifying the module saved on to the cassette.

It is good to verify so as to ensure that there is no corruption of the module during the saving procedure.

Load machine code module from cassette.

Input memory address to which the module is start loading. The address is limited to same range as in write command

b. Enter the name used when the module is saved if you are not sure of the name, just press (ENTER).

4. LIST

Display memory contents starting from an address

Procedure

a Input address start listing from.
Can be any address as in SAVE command abov
Type any key to continue the display.

c. Type "m" to return to main menu

Move memory contents from start address to finish address into new memory address.

Procedure

Input move from memory, any address as in the range of SAVE command.

Input move until memory, any address as in the range of SAVE command.

c. Input move to memory, address range as in WRITE command

You can ever copy the ROM into RAM by using this command

eg. Move from memory? 0 ENTER)
    Move until memory 1000(ENTER
    Move to memory 32000(ENTER

this will move ROM 0 to 1000 to RAM address 32000.

NOTES: Any of the input in above commands which breaches
       the address range will result in the input be
       reprompted.

EXERCISE

Try using this monitor to input the module we have developed with EZ-code.

HEXLOAD


```
100 REM
110 REM monitor program
120 CLEAR 26999 : LET ze = PI - PI
    : LET on = PI / PI : LET tw = on+on
    : LET qk = 256 : LET lm = 27000
    : LET mr = 140 : LET wl = 340
130 GO SUB 2000
140 CLS
    : PRINT "Start of machine code area = "
      ; lm
150 PRINT "menu" : PRINT
    : PRINT
      "    Write machine code..
160 PRINT
    : PRINT
      "    Save  machine cod
170 PRINT
    : PRINT
      "    Load  machine code......
180 PRINT
    : PRINT
      "    List  machine code..
190 PRINT
    : PRINT
      "    Move  machine code...     ...."
[illegible]
      [illegible] appropriate key."
[illegible]
220 IF g$ = "m" OR g$ = "M" THEN STOP
230 IF g$ = "" OR g$ < "1" OR g$ > "5"
      THEN GO TO 210
240 CLS
    : PRINT "Start of machin  code area = "
      ; lm
250 GO TO 300* VAL g$
300 REM  [illegible] Write******************** [illegible]
310 INPUT "Write to address ?";d
320 IF d > mm OR d < lm THEN GO TO 310
330 PRINT : PRINT "Write Address ? ";d
    : PRINT "To return to menu enter ""m"""
340 LET a$ = ""
[illegible] $ = "" THEN INPUT "Enter hex code "
      ; a$
360 IF a$ (on) = "m" OR a$ (on) = "M"
      THEN GO TO mr
370 IF LEN a$ / tw <> INT ( LEN a$ / tw)
      THEN PRINT "Incorrect entry ";
    : GO TO wl
```

```
380 LET c = ze
390 FOR f = 16 TO on  STEP  15
400 LET a = CODE a$((f = 16)+tw*(f = on))
410 IF a < 48 OR a > 102 OR (a > 57 AND a < 65)
       OR (a > 70 AND a < 97)
       THEN PRINT "Incorrect▲entry▲";
    : GO TO wi
420 LET c = c+f*((a < 58)*(a-48)
       +(a > 64 AND a < 71)*(a-55)+(a > 96)*(a-87))
430 NEXT f : POKE d, c : LET d = d+on
440 PRINT a$( TO tw); "▲▲";
450 LET a$ = a$(3 TO )
460 IF d = UDG
       THEN PRINT
       "Warning :you▲are▲now▲in▲the▲user
       graphics▲area!"
    : GO TO wi
470 IF d = UDG-20
       THEN PRINT
       "Warning :you▲are▲now▲in▲routines
       memory▲area"
         : GO TO wi
480 GO TO wi
600 REM  IBL Save******************** IBL
610 INPUT "Save▲M.C.▲from▲address▲: "; a
620 INPUT "Number▲of▲bytes▲to▲be▲saved▲: "; n
630 INPUT "Name▲of▲the▲routine :"; a$
640 SAVE a$ CODE a, n
650 PRINT "Do▲you▲wish▲to▲verify?"
660 INPUT v$
670 IF v$ <> "y" THEN GO TO mr
680 PRINT "Rewind▲tape▲and▲press▲""PLAY"""
690 VERIFY a$ CODE a, n
700 PRINT "O.K." : PAUSE 50
710 GO TO mr
900 REM  IBL Load******************** IBL
910 INPUT
         "Load▲M.C.▲to▲address▲starting▲▲▲ :▲"
         ; a
920 IF a > mm OR a < lm THEN GO TO 910
930 INPUT "Program▲name▲: "; a$
940 PRINT "Press▲""PLAY""▲on▲tape."
950 LOAD a$ CODE a : GO TO mr
1200 REM  IBL List******************** IBL
1210 LET a$ = "0123456789ABCDEF"
1220 INPUT "List▲Address :"; d
1230 PRINT "Press▲""M""▲to▲return▲to▲Menu."
1240 LET a = INT ( PEEK d/16)
     : LET b = PEEK d-16* INT ( PEEK d/16)
1250 PRINT d; TAB 7; a$(a+on); a$(b+on)
1260 LET d = d+on
1270 IF INKEY$ = "m" OR INKEY$ = "M" THEN GO TO mr
1280 GO TO 1240
1500 REM  IBL Move******************* IBL
```

```
[illegible]1  INPUT "Move from memory #: "; fm
[illegible]  INPUT "Move until memory #: "; um
[illegible]  INPUT "Move to memory #: "; tm
[illegible]  IF tm > fm THEN GO TO 1610
[illegible]  LET [illegible] = [illegible]
[illegible]  FOR [illegible] = fm TO [illegible]
     [illegible]
[illegible]  LET [illegible] = [illegible]
[illegible]  [illegible]
[illegible]  [illegible]
     [illegible] = [illegible] fm
[illegible]  FOR [illegible] = um TO fm STEP -[illegible]
[illegible]  [illegible] PEEK [illegible]
[illegible]  LET [illegible] = [illegible]
[illegible]  [illegible]
     [illegible]
     LET [illegible] = PEEK [illegible] PEEK [illegible]
     [illegible] = 65535 THEN LET mm = 65 [illegible]
     [illegible]
[illegible]  [illegible] = [illegible]67 THEN LET mm = 325 [illegible]
     [illegible]
[illegible]  [illegible]
[illegible]  [illegible]
[illegible]  [illegible]
```

The Freeway Frog Program

Program Design

[illegible]
one side of a highway to the other.

[illegible]
police cars frequently patrolling the highway.

[illegible]
other si[illegible]

[illegible]
programmer.
This is merely the problem definition [illegible]
[illegible]
the design and development of the whole project.

FREEWAY FROG program structure

[illegible]
program up into well-defined logical modules

They are as follows.

1. INITIALISATION
   perform all initial tasks.

2. TRAFFIC FLOW
   control of traffic on the highway.
   This can again be logically subdivided into
   i. regular traffic flow eg. trucks, cars and motorcycles.
   ii. irregular flow traffic eg. police car.

3. FROG
   control the movement of the FROG, crash testing as well as home testing.

4. GENERAL PROGRAM CONTROL
   this part of the program takes care of the score calculation and display, testing for termination of the game.

5. TERMINATION
perform the house keeping job before returning from the program.

Developing the FREEWAY FROG program

[illegible] the logical breaks shown above.

[illegible] stage is working before proceeding to the next stage.

The six stages will be

1. Data Base design
involving the design of objects shape, the creation of [illegible]

2. Initialisation
involves the setting up of the screen, and the initialisation of various variables.

3. Traffic flow
[illegible] different logic.

4. Police car
we develop and t t the police car movement.

5. Frog
[illegible] crashing, calculating scores ...etc.

6. Program control
[illegible] game, return from the program

[illegible] block of memory and generate the sum as a "checksum".

[illegible]

```
9000    REM
9010    REM checksum
```

```
9020    INPUT "From address
9030    INPUT "To address
[illegible]
9050    FOR I=f TO t
9060    LET s=s+PEEK I
9070    NEXT I
9080    PRINT "Checksum  "
9090    GO TO 9020
```

[illegible] block, [illegible] end of the [illegible]
block which you want to do the checksum in decimal value.
The BASIC program will generate the checksum valu[illegible]

Stage 1 - Data Base

```
****  Design of object shape  ****
```

As this is a two way traffic game, we need to d[illegible]ign two truck shapes a left truck shape and a right truck shape etc...

[illegible] will be four shapes, one for each direction.

[illegible] for drawing each object:

If the shape is composed of four characters

```
A  B
```

the position pointer will be pointing to character A.

Character A [illegible] drawn first, then character B ...until the whole row is finished. [illegible] above to character C

Thus, we will organise the shape data [illegible].

```
Shape   AB[illegible]D
```

[illegible]

[illegible] will look like this

```
Shape   a1, a2, a3, a4, a5, a6, a7, a8
        b1  b2  b3, b4, b5, b6, b7, b8
                c3, c4, c5, c6, c7, c8
                d3, d4, d5, d6, d7, d8
```

[illegible] change the attribute file.

[illegible] shape after its screen memory data

Unlike the shape, for each character there is only one corresponding attribute data byte.

So, to cater for the attributes data we have four attribute data bytes after the above thirty two shape data bytes. (for a four character shape)

**** Input of object shape ****

Label	line#	from(H)	to(H)	from(D)	to(D)
FRGSHP	120			27055	27190
LBIKE	340	6A37H	6A76H	27191	27254
LBATT	430	6A77H	6A7EH	27255	27262
RBIKE	460	6A7FH	6ABEH	27263	27326
RBATT	560	6ABFH	6AC6H	27327	27334
LCAR	610	6AC7H	6B26H	27335	27430
LCATT	730	6B27H	6B32H	27431	27442
RCAR	770	6B33H	6B92H	27443	27538
RCATT	900	6B93H	6B9EH	27539	27550
LTRUCK	940	6B9FH	6C76H	27551	27766
LTATT	1230	6C77H	6C91H	27767	27793
RTRUCK	1280	6C92H	6D69H	27794	28009
RTATT	1570	6D6AH	6D84H	28010	28036
BLANK	1620	6D85H	6D88H	28037	28040

Module from 27055 to 28040, 986 bytes, checksum is 79197.

Suggested name "shapdb",(shape database).

data bytes followed by attribute data bytes.

one time, either GREEN when it is alive, or RED when it is dying or YELLOW when it reaches home.

(7) as paper colour.

0 and the ink colour will be that given in its databa

cassette, it is assumed that you understand charact representation in memory

shape FROG1, starting at line 160.

In line 160, you will see

```
6987    6F  160 FROG1   DB  111,15,31,159,220,216,120,48
        0F 1F 9F DC D8 78 30
```

69B7 is the memory address in hexadecimal format

6F is the start of the eight bytes of the current DB instruction in Hexadecimal value.
The hexadecimal value of the next seven bytes [illegible] in the next line between line 160 and line 170, ie 0FH, 1FH, 9FH, DCH, D8H, 78H, 30H.

160 is the line number of the assembler listing.

FROG1 is the label. This is for our benefit only.

DB is a mnemonic. It means that what follows is a [illegible]nce of bytes. (Similar to DATA in BASIC).

,111,15,31,159,220,216,120,48
are the bytes to be loaded into the memory.

Now let's build the FROG1 sh[illegible]

00	00000000	00000000	[illegible]
01	00000001	10000000	[illegible]
23	00100011	11000100	
25	00100101	10100100	[illegible]
6F	01101111	11110110	[illegible]
4F	01001111	11110010	
DF	11011111	11111011	F[illegible]
FF	11111111	11111111	[illegible]
6F	01101111	11110110	F6
0F	00001111	11110000	F[illegible]
1F	00011111	11111000	F8
9F	10011111	11111001	F9
DC	11011100	00111011	3B
D8	11011000	00011011	1B
78	01111000	00011110	1E
30	00110000	00001100	0C

Remember,
i. we draw the bottom row first from left to right.
ii. Then we draw the next row up.
iii. For each character, we draw the eight bytes from top to bottom
iv. Then at the very l[illegible], we fill in the attributes.

to find the correct shape given the direction of the frog.

DEFW is a mnemonic that means we want to define a 2-byte "nn". The least significant byte is first while the most significant byte is next.

**** Input of shape database ****

[illegible] listing. Enter only the hex bytes as shown in column 2.

[illegible] part of this stage'

**** Design of the objects database ****

[illegible] in the two lane of the highway
These are randomly distributed between the two lanes

Object database will store information about the current status of the traffic

For example, for each object we need to know:

[illegible] it's partly on the screen or not, Position pointer.
Shape database pointer, Attribute database pointer,
Number of Rows the shape occupies,
Number of column the shape occupies.

The dat[illegible] carries this information about each object in each game cylce.

[illegible] generated randomly

One simple way is to prepare the initial information for each possible vehicle and store this in memory.
[illegible] memory locations and restore the da[illegible]

[illegible]

[illegible] progr[illegible]

* temporary database memory map

Existence	DEFB	1 byte
Cycle count	DEFB	1 byte
Direction	DEFB	1 byte
Real/abstract	DEFB	1 byte
Position	DEFW	2 bytes
Shape pointer	DEFW	2 bytes
Attribute	DEFW	2 bytes
Row	DEFB	1 byte
Column	DEFB	1 byte
	TOTAL	12 bytes

Label	line#	from(H)	to H,	from(D)	to(D
OB1EXT	1710	6E25H	6E30H	28197	28208
OB2EXT	1800	6E31H	6E3CH	28109	28220
OB3EXT	1850	6E3DH	6E48H	28221	28232
OB4EXT	1900	6E49H	6E54H	28233	28244
OB5EXT	1950	6E55H	6E60H	28245	28256
OB6EXT	2000	6E61H	6E6CH	28257	28268
PCAREXT	2070	6E6DH	6E78H	28269	2828[illegible]
FROEXT	2180	6E79H	6E80H	28281	28.88

As mentioned above, these are only temporary working stor
information that they contain changes as the game proceed

There are two other major temporary working storage area

car respectively.

Label	line#	from H)	to(H)	From D)	to D
FRGSTR	1650	6D89H	6DACH	28041	28076
PCSTR	1660	6DADH	6E24H	28077	28196

them. We only need to build up the following database.

The object database is organised in the following wa

	right bycycle datab
	left bycycle database
	right car database
	eft car database
[illegible]	right truck database
LTDB	left truck database

LPCDB	left police car database
LPCATT	left police attribute da .
RPCDB	right police car database
RPCATT	right police car database

Label	Line#	from(H)	to(H)	from(D)	to(D)
FRGDB	2260	6E81H	6E88H	28289	28296
DBINDEX	2320	6E89H	6E94H	28297	28308
RMDB	2400	6E95H	6EA0H	28309	28320
LMDB	2470	6EA1H	6EACH	28321	28332
RCDB	2540	6EADH	6EB8H	28333	28344
LCDB	2610	6EB9H	6EC4H	28345	28356
RTDB	2680	6EC5H	6ED0H	28357	28368
LTDB	2750	6ED1H	6EDCH	28369	28380
LPCDB	2820	6EDDH	6EE8H	28381	28392
LPCATT	2890	6EE9H	6EF4H	28393	28404
RPCDB	2930	6EF5H	6F00H	28405	28416
RPCATT	3000	6F01H	6F0CH	28417	28428

Module from 28289 to 28428, 140 bytes, checksum 7697.

Suggested name is "objdb". (object database).

[illegible]
database.

The meaning and contents of each byte is.

* Existence (1 byte)
  - set to zero when the object is nonexistant.
  - set to value n where (n - 1) is the number of cycles
    that the object will wait before it is allowed to move

n value for	left and right cycle	is	2
	left and right car	is	3
	left and right truck	is	6
	police car	is	1
	frog	is	0

    in other words, the police car moves every cycle,
    the motorcycle move every alternate cycle etc.

* Cycle count (1 byte)
  - initially set as 1 so that it is ready to move straight
    away and decrement by one every cycle.
  - when it reaches zero, the object will be allowed to
    move and the count will be reinitialised
    to the value held in the existence byte

* Direction (1 byte)
  - all left to right traffic (ie. top lane traffic) will
    have direction value zero.

- all right to left traffic (ie bottom lane traffic)
  will have direction value one.

* Abstract/Real Flag (1 byte)
- this defines whether objects is partly off the screen
- all left to right traffic will start off with value
  zero (abstract).
- left to right traffic will change this to one when
  their position points to the real screen 4820H.
  all right to left traffic will have flag start off with
  value one (real); the object has a position pointing to
  the screen, ie 48DFH
- as the right to left traffic moves off the screen,
  ie,when the position pointer moves from 48C0H to 48BFH,
  this will be changed from real to abstract.

* Position pointer (2 bytes)
- 2 bytes pointer storing the current position of the
  object.

* Shape pointer (2 bytes)
- 2 bytes pointer pointing to the shape datab[illegible] of the
  object.

* Attribute pointer (2 bytes)
- 2 bytes pointer pointing to the attribute data[illegible]e of
  the object.

* Row (1 byte)
- store how many rows the object shape occupies.

* Column (1 byte)
- store how many columns the object shape tak[illegible]
- this value includes two columns of blanks, one at each
  end of the object.
  The purpose of these two extra columns of blanks is to
  avoid the traffic getting too close to each other.

Now you can key in the object initialise data[illegible]e from listing 2270
to 30[illegible]0.

You can use EZ-code or HexLoad to enter this module.
If you use EZ-code, remember to save the source listing [illegible]
the dumped listing.

**** General database ****

We have covered so far the database from 69AFH to 6F0CH
(27055 to 28428).

Now we are going to build up the rest of the database and we

classify this as "general database'.

This is organised as below

line	500 to 650	SOUND	
	660 to 690	SCORE MESSAGE	
	720 to 1210	GENERAL	

label	line#	from H)	to,H)	from D)	to D)	checksum
PCTON1	500	6F0DH	6F10H	28429	28432	282
PCTON2	510	6F11H	6F14H	28433	28436	166
HOMTON	540	6F15H	6F3CH	28437	28476	2565
SCRMS1	660	6F3DH	6F42H	28477	28482	540
SCORE	670	6F43H	6F48H	28483	28488	288
SCRMS2	680	6F49H	6F53H	28489	28499	732
HISCR	690	6F54H	6F58H	28500	28504	240

Module from 28429 to 28504, 76 bytes, checksum 4813.

Suggested name ' [illegible] general [illegible] e).

You only need to input from line 500 to line 690.

From line 720 to line 1210, memory 6F59H to 6F82H (28505 to 28546), these are all variables used by the progr

Line 1100 to 1150 are instructions with mnemonic EQU. This [illegible] a value to the corresponding label and is used by the [illegible] program. You do not have to enter anything

onclusion

Now we have covered the whole database a[illegible] from memory 69AFH TO 6F82H (27055 to 28546).

[illegible]

FREEWAY FROG program.

You should have now developed three modules.

name	from mem	to mem	length	checksum
shpdb	27055	28040	[illegible]	
objdb	28289	28428		[illegible]
gendb	28429	28504	76	[illegible]

Note that the database occupies nearly 1400 bytes''

Stage 2 - Initialisation

**** Screen Setup ****

In this module, we set up the highway, the score display, the frog as well as initialise all control variab..

We will do it in three parts.
First, clear the screen and put in the highway.
Secondly, put in all the frog
Thirdly, display the score.

This module includes the following routine

routine	line#	from(H)	to(H)	from(D)	to(D)
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
DAWHWY	1820	7000H	7040H	28683	28736
FILHWY	2070	7041H	7054H	28737	28756
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Module spread from 28547 to [illegible]744, 2201 bytes.

Suggested name "init". (initialisation).

[illegible]

memory locations.

Then enter INIT routine. Enter three bytes of zero for the [illegible]

called haven't been developed ye[illegible]

line#	address(H)	address(D)
1430	6FAFH	[illegible]
1470	6FBAH	[illegible]
[illegible]490	6FC[illegible]H	[illegible]
1530	6FCPH	[illegible]
1570	6FD[illegible]H	[illegible]
1590	6FD[illegible]H	[illegible]
1630	6FE7H	[illegible]

[illegible]

code in memory 32000.

```
F3             DI                        ;Disable interrupt
D9             EXX                       ;Preserve HL'
E5             PUSH    HL
D9             EXX
CD8[illegible]0F         CALL    INIT
3E7F    KEY    LD      A,7FH             ;TRAP SPACE KEY
DBFE           IN      A,(FEH)
E601           AND     1

CDFE77         CALL    FINAL             ;finalisation
D9             EXX                       ;restore HL'
E1             POP     HL
D9             EXX
FB             EI                        ;enable interrupt
C9             RET
```

You should see the screen blacken and four white lines

```
INIT
    set border colour to black
    initialise frog-crash flag, frog existence, gameflag
               number of frog
      set random ROM pointer
      set frog station (also initial position of frog) to
                "A"
    call clear screen
    call draw highway
    call line-up fr[illegible] (five of them)
    load score message
    print score
    load high score message
    print high score
    initialise all objects as nonexistent
    initialise chase flag, siren sound flag and score


DRWHWY
    fill top highway line (32 characters of 40A0H)
    fill middle highway line (32 characters of 4860H)
    fill bottom highway line (32 characters of 5020H)
*remember that highway is white paper black ink
    unfill top two-character bytes of top highway
                               (therefore, they are white)
    unfill bottom two-character bytes of bottom highway
                                they are also white now)
    redraw middle two-byt[illegible] of the middle highway
```

```
FILHWY
    initialise fill character ()FFH
    set loop count to 32       (one line 32 characters)
    draw one character (8 bytes)
    move pointer to next character each time

FINAL
    set white border
    blank screen
    set screen attribute file to white paper and black ink
```

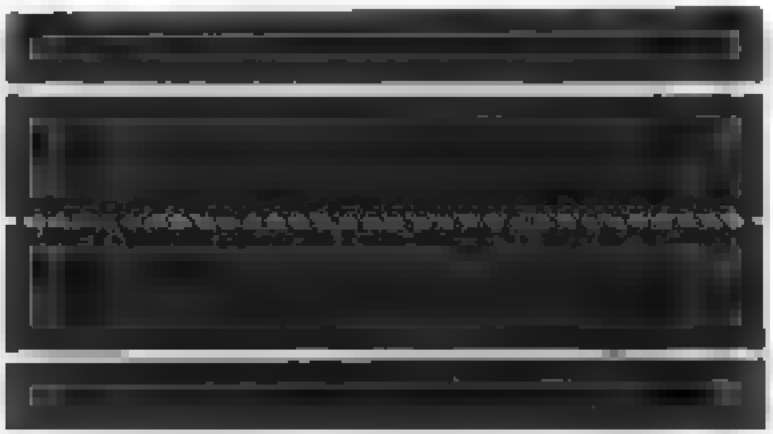
If everything is fine, save the module first from memory location 28500 to 30800, 2300 bytes.

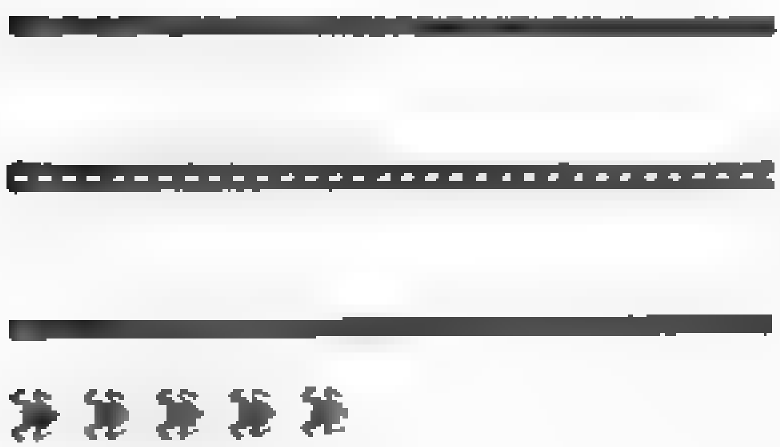
Now enter LINEUP, DRWFRG routines. Check the checksum and save the whole module again under the same name, same addresses. Then change memory from 6FAFH ([illegible]) to 6FB1H ([illegible]) so that it corresponds to line 1430 of the assembly listing ie. CD 55 70.

Run the program and you will see five frogs lined up at the left bottom of the screen.

The following are descriptions of what the two routines do.

```
LINEUP
    set frog direction to 1 (facing right)
    set frog shape to FROG2
    set attribute number to 2 (green)
    if no frog left
        then return
    else
        for number of frog
            push BC, DE, HL onto stack
            draw the frog by calling DRWFRG routine
            pop  HL, DE, BC from stack
            update draw position


DRWFRG
    draw shape using convention discussed earlier
    calculate attribute pointer
    fill attribute of frog
```





Stage 3 – Regular Traffic

In this stage, we develop the regular flow of traffic. ie. all traffic except police car

```
    Traffic control ( including regeneration of traffic)
        regenerate traffic
    Moving traffic
        moving control
              awing traffic
                determine drawn shape
```

Below is a table of all routines in this module.

name	line#	from(H)	to(H)	from(D)	to(D)	[illegible]
TFCTRL	3090	70BDH	70D8H	2886[illegible]	[illegible]	[illegible]
REGEN	3320	70D9H	710EH	28889	[illegible]	[illegible]
MVTRF	3700	710FH	71AEH	28943	[illegible]	[illegible]
MVCTRL	4720	71AFH	7208H	29103	[illegible]	[illegible]
DRAW	5560	7209H	7295H	29193	[illegible]	[illegible]
DSHAPE	6630	7296H	72D6H	29334	[illegible]	[illegible]
RANDNO	15050	77CCH	77D[illegible]H	30668	[illegible]	[illegible]

Module from 28861 to 30685, 1824 bytes.

Suggested name "regtrf". (regular traffic).

[illegible] It is useless to generate a total checksum of the whole [illegible] use the memory range covers some undeveloped memory [illegible] routine after entering it.

We develop this module in two parts.

Firstly, the draw routine for traffic.
Secondly, the traffic control and draw control.

[illegible] save them

[illegible] testing program starting from memory 32000.

```
    F3                DI
    D9                EXX
    E5                PUSH    HL
    D9                EXX
    CD836F            CALL    INIT
    3E03              LD      A, 3          ;row count
    [illegible]       [illegible] [illegible] A  ;[illegible]
    [illegible]               A              [illegible]
    325F6F            LD      (COLUMN), A   ,store in COLUMN
    11926C            LD      DE, RTRUCK    ;right truck shape
    216A6D            LD      HL, RTATT     ,right truck attrl
    226A6F            LD      (ATTPTR), HL,store in ATTPTR
    3E01              LD      A, 1          ;set to real pos
    212248            LD      HL, 4822H     [illegible]
    CD0972            CALL    DRAW          [illegible]
    3E7F      KEY     LD      A, 7FH        ;key trap
    DBFE              IN      A,(OFEH)
    E601              AND     1
    20F8              JR      NZ, KEY
    [illegible]       CALL    FINAL
    D9                EXX
    E1                POP     HL
    D9                EXX
    FB                EI
    C9                RET
```

[illegible] modules in the order they are created
Load the init module.
Load the routines you developed in this stage
[illegible]
includes all routines you have developed so far.
Enter and save the above test routine in memory 32000.

[illegible]
the top lane as well.

[illegible]
INIT and CALL DRAW to test all other object shapes

Below is a brief description of the two routines.

DRAW
    Similar logic to DRWFRG

RSHAPE
    trap lower 5 bits of low order byte of position
    parameter
    subtract from 1FH and add 1

trap lower 5 bits again
determine SKIP and FILL depending on real or abstract
calculate attribute position and store in ATTPOS

save the whole module.

Edit the testing routine as follows

```
                        EXX
                        PUSH    HL
                        EXX
                        CALL    INIT
CDBD70   MOVE           CALL    TFCTR
CDOF71                  CALL    MOVTRF
3E7F                    LD      A, 7FH
DBFE                    IN      A,(OFEH)
E601                    AND     1
20F2                    JR      NZ, MOVE
                        CALL    FINAL
                        EXX
                        POP     HL
                        EXX
                        EI
                        RET
```

we are developing that stage
Once a module is fully developed and tested, it will be merged

12000

test run the new "frog" module.

plus all traffic moving at a very fast speed in the two lanes. This is because there is no delay between each program cycle.

A short description of the routine

TFCTR
    load generation flag
    if not regeneration
        decrement flag count

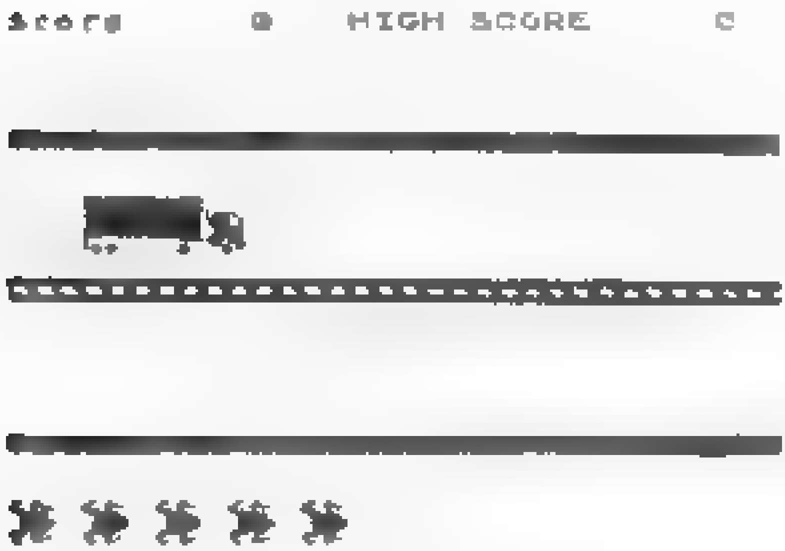
    else
        regenerate the first nonexistence object by calling

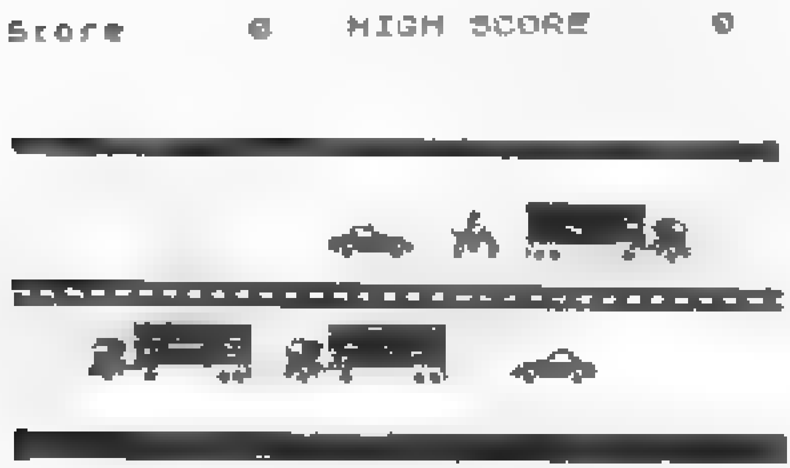
```
        the REGEN routine
    return


REGEN
    save existence database pointer
    generate random number 0 to 5
    test the first two character of the screen position
        where the object is created
    if the sum of the attributes of those two position is
        not equal to zero
            then return (traffic jam)
    else
        determine the initialise database
        load into temporary working databa
        set regeneration cycle count to 2
        return


MVTRF
    for all existing objects
        decrement cycle count
        if count reaches zero
            reload count from existence
            move one character left or right
            store new position in NEWPOS
            test attribute correspondence of the front of
                objects
            if any nonzero ink
                if not green
                    set jam flag
                else
                    set crash flag
            if jam flag set
                load cycle count with 2(move one cycle
                                            later)
                return
            else
                store new position
                call MVCTRL (move control


MVCTRL
    if edge reached
        change real/abstract flag
    if left moving
        if on edge (position low order byte = 1FH)
            if abstract flag
                set non exist, return
            else
                goto L1
        else
```

nter in ATTPTR

LUMN of shap

what random pointer pointing to in ROM





Stage 4 - Police Car

In this stage we will introduce the POLICE car into the program.

The Police car will be generated randomly and enter with a SIREN sounding [illegible] traffic jam [illegible] course. It will overtake any regular traffic before it.

That is why the program needs to save what is underneath the police car and put it back when the police car moves on.

Below are all routines in this module.

name	line#	from(H)	to(H)	from(D)	to(D)	checksum
RESPC	9560	7430H	74C1H	29776	29889	11011
POLICE	7930	734AH	73DEH	29514	29662	15769
STRPC	8830	73DFH	744FH	29663	29775	10615

[illegible] developed in previous modules.

Module from 29514 to 29889, 376 bytes

Suggested name "police".

[illegible] check [illegible] and save them on the "police" module.

Then edit the testing program to the following

```
                      EXX
                      PUSH    HL
                      EXX
                      CALL    INIT
              MOVE    CALL    TFCTRL
                      CALL    MOVTRF
   CD4A73     MOVE1   CALL    POLICE
                      LD      A,7FH
                      IN      A,(0FEH)
                      AND     1
   20F5               JR      NZ,MOVE1
                      CALL    FINAL
                      EXX
                      POP     HL
                      EXX
                      EI
                      RET
```

Load the "frog" module, then the "police" module.

[illegible]
fast on the highway.

If you want to put in other traffic as well, change the relative jump to JR NZ, MOVE
Remember to recalculate the displacement offset. (EFH).

[illegible]
shape as it overtakes them. It may be so fast that you wouldn't notice.
[illegible]
in front of them.

Let's look at these routines

```
POLICE
    if police car non-exist
        get a random number
        if not a mutiple of 3,
            return
        else
            set chase flag
            determine top or bottom lane randomly
            load corresponding initial data
    set direction
    store position pointer
    retrieve position
    move and store NEWPOS
    set ROW, COLUMN, REAL/ABSTRACT flag, POS before
        call RSHAPE
    get resulted ATTPOS and test head of shape for green
    if green attribute
        set crash flag
        blank front of police car
    call STRPC (for storing of what's underneath policecar)
    update position database
    call MVCTRL (for moving on and off the screen)
    turn off chase flag if non-exist


STRPC
    set HL points to NEWPOS
    set DE points to PCSTR (police car store)
    store position and 5 byte of information starting from
        R-W variable
    store according to SKIP/FILL format
        all screen memory first
        then attribute file
```

Save the whole module as "police"

You need to edit the testing program again to test the storing and restoring

Although we included the STRPC routine we are not sure that it

the screen

change the tesing program to the following

```
        MOVE    CALL    TECTRL
CD5074          CALL    RESPC
                CALL    MOVTRF
                CALL    POLICE
                LD      A,7FH
                IN      A,(0FEH)
                AND     1
20EC            JR      NZ,MOVE
```

Adjust the relative jump before you test run the module with 'frog'

them off

The RESPC's logic is as below

RESPC
- return if police car nonexist
- restore the position and 5 bytes into variables starting from ROW
- restore screen memory then attribute according to SKIP FILL format
- return

of the "police" module.

Load "frog" module then reload the new "police" module.

Edit the testing program in memory 32000's following

```
                 N T
                 TRCTR
                 RESP
                 MOVTRF
                 POLICE
D877             SERFN
                 A,(OFFH)
                 MOVE
```

... will find the whole ... This ... constant delay ... downcount delay loop

... press ...

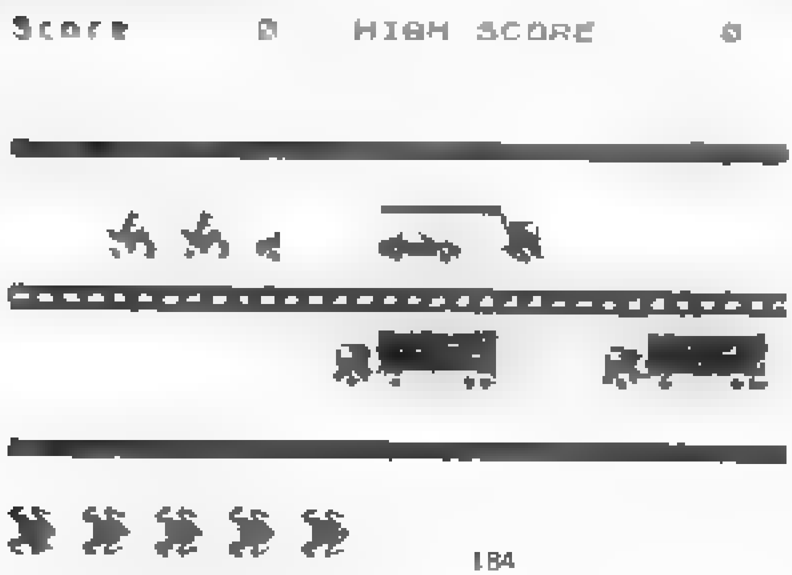
go to DELAY

if no police

... DE, H

... 038

... "frog" in the memory and ...



Stage 5 – The Frog

In this stage we develop the frog routines.

We need to regenerate the frog when a frog dies.

We also need to handle the frog movement, save what [illegible] underneath it and restore what was stored back when the frog moves.

We also need to handle the frog crashing, [illegible] home run [illegible]

We build up this module in three parts as below.

- Regenerate and move frog
- Store and restore what is underneath
- handle crashing, homerun and score

All routines within this module is as follows.

name	line#	from(H)	to(H)	from(D)	to(D)	checksum
FROG	10280	74C2H	74E2H	29890	29922	3818
REGFRG	10320	74E3H	750FH	29923	29967	4079
MOVFRG	10770	7510H	75D5H	29968	30165	19943
RESFRG	11870	75D6H	7627H	30166	30247	8492
STRFRG	12440	7628H	7690H	30248	30352	10136
CRASH	13160	7691H	76A6H	30353	30374	2767
FRGDIE	13280	76A7H	7707H	30375	30471	9965
FRGTON	13890	7708H	771CH	30472	30492	2435
CALSCR	14040	771DH	776EH	30493	30574	8106

Module from 29890 to 30574, 685 bytes

Suggested name "frgrtn" (frog routine).

[illegible]

Edit the testing program as following

```
          CALL   INIT
MOVE      CALL   TFCTRL
          CALL   RESPC
          [illegible]
```

```
        LD      A,7FH
        IN      A,(OFEH)
        AND     1
        JR      NZ,MOVF
```

in line 13190 of the assembly listing by
        00, 00, 00

Run the testing progr   nd you should be able to move the frog. The controls are "1"=up, a" down, "i"=left, "p"=right.

put in yet.

The description of these few routines are as following

FROG
 the control routine for the whole FROG module

```
    if frog crashed
        goto CRH

          t score-flag to no score (0)
        call RE FR
        decrement cycle count
        if count non zero
            return
        else
            reset cycle count
            call MOVFR
            if not crash
                return
CRH call CRASH routine
    return
```

REFRG

```
    if frog does not exist
        load frog initial database to working database
        update frog station to three position left
        initialise OLDFRG and NEWFRG to FRGPOS
        initialise frog storage area to 0
```

MOVFR

```
    initialise registers
                C   absolute movement
                B - frog direction
```

```
                DE- frog shape
    test frog movement
                1   up, a - down, i - left, p - right
    store shape, direction
    if absolute move is zero
        return
    else
        restore old frog position
        calculate new frog position and store
    test up screen position, right screen, bottom screen,
        frog station
    if valid
        store position into NEWFRG
        set score flag.
    restore OLDFRG
    if OLDFRG equal NEWFRG
        return

        call RESFRG
        set OLDFRG equal NEWFRG
        move stored direction, shape pointer to frog
            database
        call STRFRG
        return


RESFRG
    restore underneath frog based on OLDFRG position
    memory first then attributes


STRFRG
    store underneath frog based on NEWFRG position
    draw frog while drawing


CRASH
    reset crash flag
    set frog nonexist
    call FRGDIE routine (dying procedures)
    call RESFRG routine (place back what was underneath)
    decrement number of frog
```

and FRGTOM routine.

the following instruction

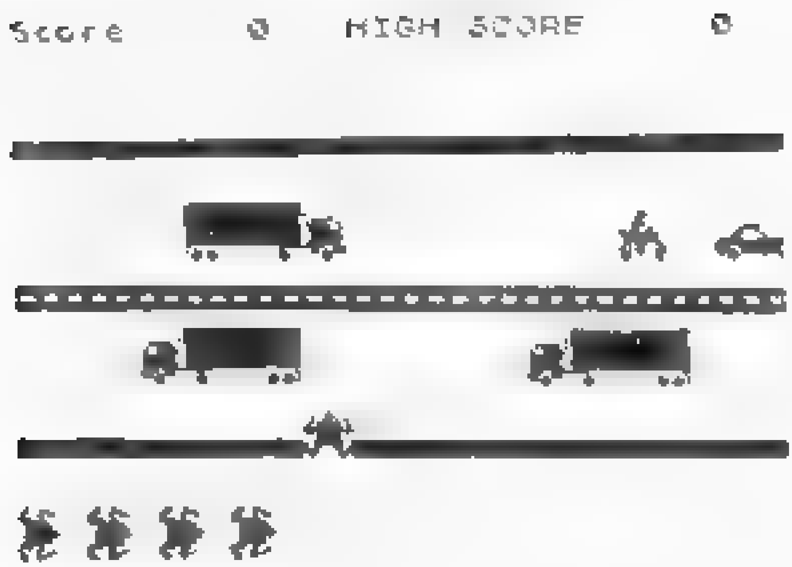
```
7698    CDA776      CALL    FRGDIE      (30)60
```

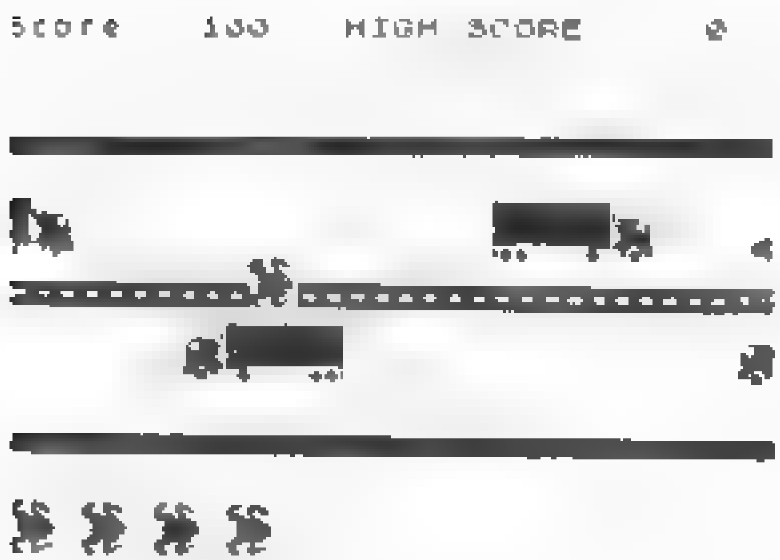
Edit the testing program to the following

together with the "frog" module

When the frog cr hes, it will flash red and vanish.

```
FR DIE
    test frog reaches home or die
      t die tone, red colour attribute
       reaches home
        add one to third digit of score
            (bonus 100 points)
        call DISSCR (display score)
        set home tone, yellow colour attribute
    draw frog based on OLDFRG, FROGSH, and attribute just
        set by call DRWFRG
        h frog with the attribute five times


FR TON
    call TONE1 (tone code from SIREN routine)
    move up or down the tone database depends upon attibute
        used to flash the frog ( if yellow then down db )
                               ( if red    then up   db )


CALSCR
    if frog non-exist
        return
    e
        if score flag not set
            return
        else
            if go up
                add one to tenth digit of score
                    (10 points)
            else
                if not within the highway
                    return
                else
                    add one to tenth digit
```


Score 0 HIGH SCORE 0


Score 100 HIGH SCORE 0

Stage 6–Control

[illegible]

[illegible] updated and the game is restarted automically.

[illegible] key.

You will find that line 180 to 440 of the listing looks very similar to the testing program we have been developing.

Routines left for program are as follows:

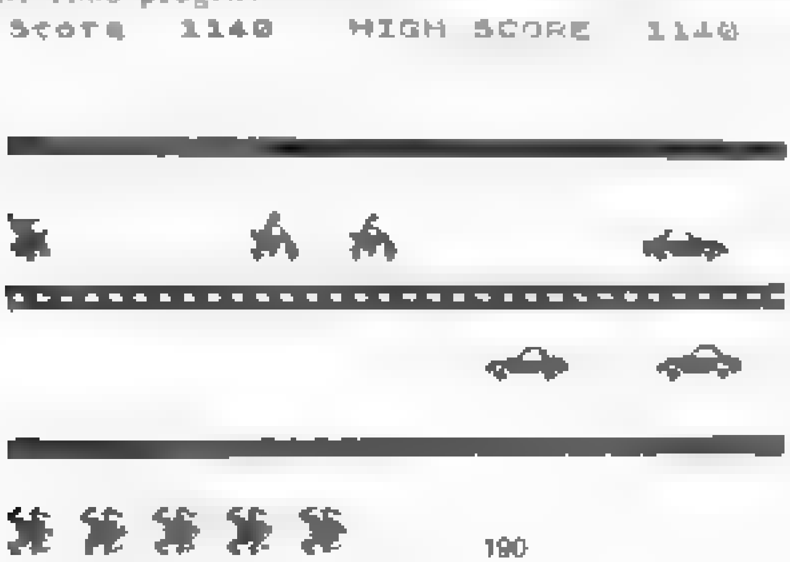
name	line#	from(H)	to(H)	from(D)	to(D)	checksum
START	180	6978H	69AEH	27000	27054	8427
OVER	13200	77DEH	77FDH	30686	30717	2491

Now [illegible] the "frog" module and save the whole module as "frog".

[illegible] working

```
OVER
    compare [illegible] l digits of HISCR and SCORE+1
    for the [illegible]rst nonequal digit
        if H[illegible]CR digit is lower
            update HISCR to SCORE+1
        else
```

[illegible] FREEWAY FROG program



```
********** FREEWAY FROG **********

                          ;PRESERVE THE HL REGISTER PAIR

                          ;INITIALISATION
                          ;TRAFFIC CONTROL ROUTINE

                POLICE    ;POLICE CAR ROUTINE
                 ROG      ;FROG MODULE
                          ;CALCULATE AND DISPLAY SCORE

                          ;HIGHSCORE MANAGEMENT

                          ;TRAP SINGLE KEY PRESSED
                                ;SCAN KEYBOARD

                          ;ENABLE INTERRUPTS
                          ;RETURN TO BASIC SYSTEM

**********                **********

             DEFM
             DEFM
             DEFM

             DB        246,240,248,249,59,27,30,12

             DB        0,1,55,57,111,79,223,255

             DB        31,31,31,127,252,193,113,56

             DB        56,113,193,252,127,31,31,31
```

```
[illegible]   [illegible]   [illegible]192,240,156,192,240,248,244,254
[illegible]   [illegible]   [illegible]255,223,28,111,37,35,1,0
[illegible]   [illegible]   [illegible]255,251,242,246,164,196,128,0
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]12,30,27,59,249,248,240,240
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]240,240,248,254,63,131,142,26
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]3[illegible],63,115,[illegible],169,112,[illegible],12,32
[illegible]   [illegible]   [illegible]254,252,252,234,213,206,14,4
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]128,192,192,224,2[illegible]4,112,119,255
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]BATT   [illegible]0,7,7,0
[illegible]   [illegible]   [illegible]0,7,7,0
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]127,63,63,87,171,115,112,32
[illegible]   [illegible]   [illegible]248,252,206,138,149,[illegible]
[illegible]   [illegible]   [illegible]
[illegible]   [illegible]   [illegible]0,0,0,0,0,0,0,0
[illegible]   [illegible]   [illegible]1,3,3,7,7,14,238,255
[illegible]   [illegible]   [illegible]128,192,128,0,192,3[illegible],[illegible]7,248
```

```
     C0 E0 00 E0 20 70 FB
6AB7 00          00530           DB      0,0,0,0,0,0,1,0

                 00540 ;

6ADF 00          00560 RBAIT     DB      0,7,7,0
     07 07 00
6AE3 00          00570           DB      0,7,7,0

                 00580 ;

6AE7 00          00600 LEAR      DB      0,0,0,0,0,0,0,0
     00 00 00 00 00 00 00

     00 07 07 0F 02 00 00
6AD7 07          00620           DB      7,255,255,159,111,247,240,96
     FF FF 9F 6F F7 F0 60
6ADF 80          00630           DB      128,255,255,255,255,254,0,0
     FF FF FF FF FE 00 00
6AE7 F0          00640           DB      240,254,255,159,111,246,240,96
     FE FF 9F 6F F6 F0 60
6AEF 00          00650           DB      0,0,0,0,0,0,0,0
     00 00 00 00 00 00 00
6AF7 00          00660           DB      0,0,0,0,0,0,0,0
     00 00 00 00 00 00 00
6AFF 00          00670           DB      0,0,0,0,0,0,0,0
     00 00 00 00 00 00 00
6B07 00          00680           DB      0,0,0,0,0,0,0,0

6B0F 00          00690           DB      0,0,0,0,0,63,97,193
     00 00 00 00 3F 61 C1
6B17 00          00700           DB      0,0,0,0,0,0,128,192
     00 00 00 00 00 80 C0
6B1F 00          00710           DB      0,0,0,0,0,0,0,0
     00 00 00 00 00 00 00
                 00720 ;
6B27 00          00730 LLAIT     DB      0,6,6,6,6,0
     06 06 06 06 00
6B2D 00          00740           DB      0,6,6,6,6,0
     06 06 06 06 00
                 00750 ;
                 00760 ;
6B33 00          00770 MEAR      DB      0,0,0,0,0,0,0,0
     00 00 00 00 00 00 00
6B3B 0F          00780           DB      15,127,255,249,246,111,15,6
     7F FF F9 F6 6F 0F 06
6B43 01          00790           DB      1,255,255,255,255,127,0,0
     FF FF FF FF 7F 00 00
6B4B             00800           DB      224,255,255,249,246,240,15,6
                 F6 EF 0F 06
                 00810           DB      0,0,0,0,0,240,64,0
     00 00 00 F0 40 00 00
                 00820           DB      0,0,0,0,0,0,0,0
     00 00 00 00 00 00 00
                 00830           DB      0,0,0,0,0,0,0,0
                 00 00 00 00
```

```
6B[illegible]  [illegible]         00840          DB    [illegible]
6B73  00                  00850          DB    0,0,0,0,0,252,134,131
      [illegible]      FC  86  8[illegible]
6B7B  00                  00860          DB    0,0,0,0,0,0,0,0
      [illegible]  00  00  [illegible]
6B[illegible]  00         00870          DB    0,0,0,0,0,0,0,0
6B8B  00                  00880          DB    0,0,0,0,0,0,0,0
      00  00  00  00  00  00  00
6B93  00                  00900 RCATT    DB    0,2,2,2,2,0
      02  02  02  02  00
6B[illegible]  00         00910          DB    0,2,2,0,0,0
      02  02  00  00  00
                          00920 [illegible]
6B[illegible]  00         00940 LTRUCK   DB    0,0,0,0,0,0,0,0
6BA7  1F                  00950          DB    [illegible]
      [illegible]  1F  3E  3D  3B  03  [illegible]
6BAF  FB                  00960          DB    [illegible]
      FC  FE  7F  BB  DB  C0  80
6BB7  FF                  00970          DB    [illegible]
      FF  FF  FF  06  [illegible]  06  06
      [illegible]
      F[illegible]  FF  [illegible]
6B[illegible]  FF         01[illegible]0          DB    [illegible]
      FF  FF  [illegible]  06  0F  [illegible]  06
6B[illegible]  FE         01[illegible]0          DB    [illegible]
      FF  FE  [illegible]  32  7A  7A  3[illegible]
6BDF  00                  01[illegible]0          DB    [illegible]
      [illegible]
6BE7  00                  01[illegible]0          DB    0,[illegible],0,0,0,0,0,0
      00  00  00  00  00  00  00
6B[illegible]  02         01050          DB    2,2,250,250,254,252,252,240
      02  FA  FA  FE  FC  FC  F0
[illegible]  FF           01060          DB    [illegible]
      FF  FF  FF  FF  FF  FF  [illegible]
6C07  FF                  01[illegible]0          DB    255,255,255,255,255,255,255,255
      FF  FF  FF  FF  FF  FF  FF
      [illegible]  FF  FF  FF  FF  FF  FF
      FF                  01090          DB    255,255,255,255,255,255,255,255
      FF  FF  FF  FF  FF  FF  FF
      FE                  01[illegible]0          DB    254,254,254,254,254,254,254,254
      FE  FE  FE  FE  FE  FE  FE
      00                  01110          DB    0,[illegible],2,0,0,0,0,0
      [illegible]
      00  00  00  00  00  00  00
      00                  01130          DB    0,0,1,0,0,0,1,0
```

```
      [illegible] 00                 00
6C3F  00
      00 00                          00
6C47  00                                        0,0,0,0,0,255,255,255
      00 00                          FF
6C4F  00                                        0,0,0,[illegible],0,255,255,255
      00 00                          FF
6C57  00                                        0,0,0,0,0,255,255,255
      00 00                          FF
6C5F  00                                        0,0,0,0,0,255,255,255
      00 00
6C67  00                                        0,[illegible],0,0,0,254,254,[illegible]
      00 00
6C6F  00                                        0,0,0,0,0,0,0,0
      00 00 00 00 00 00

                                                0,5,5,5,5,5,5,5,0
                                                0,5,5,5,5,5,5,5,0
                                                0,0,[illegible],5

                                                127,127,127,[illegible],96,96,1[illegible]
                                                255,255,255,0,96,240,240,96
         [illegible] 00
                                                255,255,255,0,0,0,0,0
         FF 00
                                                255,255,255,0,0,0,0,0
         FF 00
                                                255,255,255,[illegible],96,240,240,96
         FF FF
                                                [illegible]5,127,254,[illegible],[illegible],5[illegible]
         7F FE
                                                [illegible]40,240,240,[illegible]20,168,220,192,[illegible]
         F0 7C
                                                0,0,0,0,0,0,0,0
         00 00
                                                0,0,0,0,0,0,0,0
         00 00
                                                127,127,127,127,127,127,127,127
         7F 7F
                                                255,255,255,255,255,255,255,255
         FF FF
                                                255,255,255,255,255,255,255,255
         FF FF
                                                255,255,255,255,255,255,255,255
         FF FF
                                                255,255,255,255,255,255,255,255
            FF
                                                [illegible]
```

```
6D17 0F        01440          DB      0,0,  4,144,1,6,136, 4B, 4F

6D1A 00        01450          DB      0,1,0,0,0,0,0,0



6D32 00        01480          DB      0,0,0,0,0, 255,255,255

6D3A 00        01490          DB      0,0,0,0,0,255,255,255

6D42 00        01500          DB      0,0,0,0,0,255,255,255

6D4A 00        01510          DB      0,0,0,0,0,255,255,255
     00 00 00
6D52 00        01520          DB      0,0,0,0,0,0,0,0

6D5A 00        01530          DB      0,0,0,0,0,0,0,0

6D62 00        01540          DB      0,0,0,0,0,0,0,0


6D6A 00        01570 BIAIT    DB      0,5,5,5,5,5,5,5,0

6D73 00        01580          DB      0,5,5,5,5,5,5,5,0
     05 05 00
6D7C 00        01590          DB      0,5,5,5,5,5,0,0,0
     05 05 00

6D85 00        01620 BLANK    DB      0,0,0,0


               01640 ;
0024           01650 FRBSTR   DB      36                  ;40H+4
0078           01660 PCBIR    DB      120                 ;120H*

               ********  DATA  ********

AE 6 00        01720          DEFB    0
AE 7 00        01730          DEFB    0
AE 8 00        01740          DEFB    0
AE29 0000      01750          DEFW    0
AE2B 0000      01760          DEFW    0
AE2D 0000      01770          DEFW    0
AE2F 00        01780          DEFB    0
AE30 00        01790          DEFB    0
AE31 00        01800 DBTEXT   DB      0,0,0,0

AE35 0000      01810          DEFW    0                   ;10 *105 FROM 0AH  F A
```

```
6E39 0000        01830          DEFW     0
6E3B 00          01840          DB       0,0
     00
6E3D 00          01850 OB3EXT   DB       0,0,0,0
     00 00 00
6E41 0000        01860          DEFW
6E43 0000        01870          DEFW
6E45 0000        01880          DEFW
6E47 00          01890          DB

6E49 00          01900 OB4EXT   DB
     [illegible]
6E4D 0000        01910          DEFW
6E4F 0000        01920          DEFW
6E51 0000        01930          DEFW
6E53 00          01940          DB
     00
6E55 00          01950 OB5EXT   DB
     00 00 00
6E59 0000        01960          DEFW
6E5B 0000        01970          DEFW
6E5D 0000        01980          DEFW
6E5F 00          01990          DB
     00
6E61 00          02000 OB6EXT   DB
     00 00 00
6E65 0000        02010          DEFW     0
6E67 0000        02020          DEFW     0
6E69 0000        02030          DEFW     0
6E6B 00          02040          DB       0,0
     [illegible]

6E6D [illegible]                DB       [illegible]        ;POLICE CAR DATABASE
6E[illegible]    [illegible]
6E6F [illegible]
6E70 [illegible]                DB
6E71 [illegible]                DEFW
6E73 [illegible]                DEFW
6E75 [illegible]                DEFW
6E77 [illegible]
6E78 [illegible]                         A

6E79 [illegible]                         0                  ;FROG DATABASE
6E7A [illegible]                DB       0
6E7B [illegible]                         0                  ;0:UP 1:RGHT 2:DOWN 3:LEFT
6E7C [illegible]                DEFW
6E7E [illegible]                DEFW
6E80 [illegible]                         0

6E8[illegible] [illegible]  FROGUP  DB   [illegible]
6E[illegible] [illegible]  FROG[illegible]  DEFW  [illegible]  ;INITIAL POSITION OF FROG
[illegible]      02[illegible]  DEFW  [illegible]
```

```
6E00                                                  [illegible] CHAR?

[illegible]                                           [illegible]
[illegible]                                           ;LEFT BIKE DR
[illegible]                                           [illegible]
[illegible]                                           ;LEFT CAR DR
                                                      ;RIGHT TRUCK DR
[illegible]                                           [illegible]

[illegible]                                           [illegible]

[illegible]                                           ;POS
[illegible]                                           [illegible]
[illegible]                                           [illegible]
[illegible]                                           [illegible]
```

```
[illegible]                                DB      [illegible]

[illegible]                                DEFM    [illegible]
[illegible]                                DEFM    [illegible]
[illegible]                                DEFM    [illegible]
[illegible]                                DB      [illegible]

[illegible]                                DB      [illegible]

[illegible]                                DEFM    [illegible]
[illegible]                                DEFM    [illegible]
[illegible]                                DEFM    [illegible]
[illegible]                                DB      [illegible]

[illegible]                                DB
[illegible]                                DB

[illegible]                                DB      [illegible],1,0,0

[illegible]                                DEFM    [illegible]
[illegible]                                DEFM    [illegible]
[illegible]                                DEFM    [illegible]
[illegible]                                DB      [illegible]

[illegible]                                DB      [illegible],5,5,5,5,[illegible]
[illegible]                                DB      [illegible],5,5,0,0,0

[illegible]                                        [illegible]      [illegible]
[illegible]                                        [illegible]      [illegible]

[illegible]                                DB      [illegible]      [illegible]
[illegible]                                DB      [illegible]
[illegible]                                DB      [illegible]
[illegible]                                DB      [illegible]
[illegible]                                DB      [illegible]
```

```
        ;FROG DYING TONE REVERSE

        ;...NO SPRITE ...
        ;...DRAWN
        ;HOLDING THE ATTR ...FILE PTR

        ;NEW TRAFFIC OBJECT POSITION
        ;TRAFFIC POSITION DATABASE PTR
        ;TRAFFIC ... FLAG

        ;SET TO 1 AS TRAFFIC ...

        ;SET MAIN POLICE CAR ...
        ;SET MAIN ... SIREN SOUND
        ;DETERMINE WHICH ...
        ;POINTER TO ... FOR ...

        ;END IF ZERO
        ;OLD FROG POS
        ;NEW FROG POS
```

```
        DEFB    [illegible]
        DEFB    [illegible]
        DEFW    [illegible]
        DEFW    [illegible]

        EQU     [illegible]H    ;0,38  0,39
        EQU     [illegible]
        EQU     [illegible]     ;0,138  0,139
        EQU     [illegible]
        EQU     [illegible]
        EQU     [illegible]

        EQU     5C00H   ;FIRST 256 BYTES NOTHING

        DEFB    5       ;NUMBER OF [illegible]

        XOR     A               ;000 FOR B2 B1 B0
        OUT     (0FEH),A        ;SET BORDER COLOUR
        LD      (23624),A       ;TO BLACK
[illegible]
                                ;[illegible] POINTS TO [illegible]
[illegible]
                                ;CLEAR SCREEN ROUTINE
                                ;[illegible]
                                ;[illegible] UP ALL [illegible]
                                ;[illegible]
[illegible]
```

```
;SET ALL OBJ NONEXIST
;SET NO POLICE CAR CHASE
 SET SIREN ON
;INIT ALTSE SCORE TO
;ASCII ZERO
;FILL TOP HWY
;FILL MIDDLE HWY
;FILL BOTTOM HWY
```

```
                  INC     HL

                  RET

        ;******** ****** ********

02760 ;         draw all frogs left on the screen

                                  ;NUMBER OF FROG LIVES
                          BC
                          DE
                          HL
                          DRWFRG  ;DRAW FROG ROUTINE
                          HL
                          DE
                          HL
                          HL
                          HL
                          BC

        ;******** ****** ********

                          A,2
                          AF,AF'
                          HL      ;RETRIEVE POS PTR
                          HL
                          HL
                          A,(DE)
                          (HL),A
                          DE
                          H       ;NEXT BYTE OF THE CHAR
                          FROLP2
                          HL      ;CURRENT POINTER
                          HL      ;MOVE TO NEXT CHAR POS
                          C       ;DECR COLUMN COUNT
```

```
      20F2      02720            JR      NZ,FRGLP1
      E1        02730            POP     HL              ;ROW PTR
      08        02740            EX      AF,AF'
                                 [illegible] A           [illegible]
      [illegible]
      28[illegible]  02770       JR      Z,FRGATT        ;LOAD FROG ATTRIBUTE
                                 [illegible] AF,AF'
      A7        02790            AND     A
      ED[illegible] 02800        SBC     HL,BC           ;MOVE 32 CHAR/1 LINE UP
      CB44      02810            BIT     0,H             ;TEST CROSS SCR SECTION
      [illegible] 02820          JR      [illegible]
                                 [illegible]
                                 [illegible]             ;[illegible]
                                 [illegible] H,A
      [illegible] 02860          JR      [illegible]
      E1        02870 FRGATT     POP     HL              ;POS PTR
                                 [illegible] A,H         ;[illegible]
      E618      02890            AND     18H
      CB2F      02900            SRA     A
      CB2F      02910            SRA     A
      CB2F      02920            SRA     A
      C658      02930            ADD     A,58H
      67        02940            LD      H,A
      3A[illegible] 02950        LD      A,(ATTR)        ;FILL FROG SHAPE ATTR
      77        02960            LD      (HL),A
      23        02970            INC     HL              ;NEXT CHARACTER
      77        02980            LD      (HL),A
      ED42      02990            SBC     HL,BC           ;ONE LINE UP
      77        03000            LD      (HL),A
      2B        03010            DEC     HL              ;NEXT CHAR LEFT
      77        03020            LD      (HL),A
      C9        03030            RET
                03040 ;
                03050 ;********  TECTAL  ********
                03060 ;
                [illegible]
      [illegible]                [illegible] HL,[illegible]   ;[illegible]
      [illegible]                [illegible] A
      BE        03110            CP      (HL)
                                 [illegible]             ;[illegible]
                                 [illegible] HL          ;[illegible]
                                 [illegible]
                                 [illegible] HL,[illegible]  ;[illegible]
                                 [illegible]             ;[illegible]
                                 [illegible]             ;[illegible]
      [illegible]                [illegible]             ;[illegible]
                                 [illegible] NXSFAC[illegible]
                                 [illegible] HL,[illegible]  ;[illegible]
                                 [illegible]
                      [illegible] ADD    HL,DE
      10F[illegible]             DJNZ    [illegible]
      C9                         RET
                [illegible]
                [illegible]
                      ;********  [illegible]  ********
```

```
                ;GENERATION OF TRAFFIC
                ;TRAFFIC NEW OBJECTS

        PUSH    HL

        JR      NC,BOND1

        BIT     0,A
        JR      Z,NTHAF         ;RIGHT TRAFFIC
        LD      L,0DFH
        LD      C,0DEH

        LD      E,A

        ADD     A,(HL)
        AND     A               ;/END PAPER, ZERO IND
        JR      Z,LOADDD        ;IF 0, INITIALISE NEW OBJ
        POP     HL              ;IF SOME, RETURN

                HL,OBJECT
                DE,17
                B,
                HL

                HL              ;EXISTENCE
                A,(HL)          ;SKIP WHEN NO EXI
                A
                Z,NOTHIS
                                ;CYCLE COUNT
                (HL)            ;OF A CYCLE COUNT

                HL              ;DIRECTION
```

```
5E                [illegible]                ;POS [illegible]
23
56
1C
A7
2802
1D
1D
ED536C6F
38
010500                                       ;[illegible] DAT LENGTH

                                             ;[illegible]

                                             ;TEST DIRECTION
                                             ;RIGHT TO LEFT
                                             ;FIND END OF TRACK
                                             ;TEST [illegible]
                                               TEST [illegible]

                                             ;[illegible] TEST [illegible]
                                             ;[illegible] TO [illegible]
[illegible]
```

```
7186 ED42      04410          SBC     HL,BC
7188 08        04420          EX      AF,AF'
[illegible]
718A 20E3      04440          JR      NZ,TAKEOF
[illegible]
718F A7        04460          AND     A
7190 28[illegible]  044[illegible]0          JR      Z,MOVEOK       ;MOVE IF NO JAM
[illegible]
7194 34        04500          INC     (HL)           ;LOAD 2 TO CYCLE COUNT
[illegible]
719A 2A[illegible]  04550 MOVEOK  LD      HL,(PUSHPT)    ;HL=PIECE PTR TO POS
719D ED5B[illegible]  04560          LD      DE,(NEWPOS)
71A1 73        04570          LD      (HL),E         ;STORE NEWPOS IN DP
71A2 23        04580          INC     HL
[illegible]
      ;*********  [illegible]  ********
[illegible]
     7B        04740          LD      A,E
[illegible]
     7E        04770          LD      [illegible]
     3C        04780          INC     [illegible]
[illegible]
     77        04800          LD      [illegible]
     2B        04810 [illegible]  DEC     HL             ;PT [illegible]
[illegible]
     A7        04830          AND     [illegible]
     20[illegible]      04840          JR      NZ,[illegible]        ;RIGHT TO LEFT
[illegible]
     [illegible]        04870          JR      [illegible]
[illegible]
     7E        04890          LD      [illegible]
     2B        04900          DEC     HL             ;PT TO [illegible]
[illegible]
     2015      04920          JR      NZ,DRWOBJ
               04930          EXX
     77        04940          LD      (HL),A         ;SET NON EXISTENT
[illegible]
```

```
        DE,HL           ;TEST END OF OBJECT
        HL,BC           ;TOUCHES LEFT EDGE
        NZ,DRWOBJ
                        ;[illegible] OBJECT [illegible] LIST AS
                        ;IT MOVES OFF SCREEN
        A,(HL)
        (HL),A          ;REFILL CYCLE COUNT
        HL
        HL
        HL
        HL
        B,(HL)
        HL
        A,(HL)
        A,(HL)          ;[illegible]
```

```
                    RES
  CD9672            CALL
  3A606F            LD
  EB                EX
  D5
  E5
  5A6 6F
                                          ;SKIP POS BYTE
                                          ;MULTIPLE OF 8 BYTES

                                          ;CROSS SCREEN SECTION?
                                          ;IF YES, MOVE UP

                                          ;COLUMN TO BE FILLED
                                          ;FILL CHARACTER
                                          ;FILL CHARACTER BYTE

                                          ;NEXT CHARACTER

                                          ;RESTORE LINE PTR
                                          ;CHANGE DE PTR
                                          ;UPDATE ROW COUNT

                                          ;ONE LINE UP
                                          ;CROSS SCREEN SECTION
```

```
        ADD     A,A             ;UPDATE SHAPE DP
        EX      DE,HL
        LD      HL,(ATTROFF)
        EX      AF
        EX      DE,HL
        LD      A,(HL)
                                ;SKIP ATTRIBUTE
                                ;CLEAR CARRY
                                ;RESET ATTRIB TO LINE UP
                                ;UPDATE ATTR BYTE DP
```

```
08          06640        EX     AF,AF'          ;REAL SHAPE
7C          06660        LD     A,H
A5          06670        AND    L               ;TRAP LOWER 5 BITS
6F          06680        LD     L,A
7C          06690        LD     A,H
95          06700        SUB    L               ;SUBTRACT FROM 1FH
3C          06710        INC    A
A4          06720        AND    H               ;ADJUST FOR ZERO DIFF
6F          06730        LD     L,A
08          06740        EX     AF,AF'
                         LD
                         LD
95          06780        SUB
            06790        LD     (FILL),A
                         LD     A,L             ;RELOAD ABS DIFF
                         LD     (SKIP),A
                         CP     L               ;TAKE MIN OF
                         JR     C,RIGHTG        ;IF LL MORE THAN COR
                         PUSH   HL              ;CALCULATE ATT PTR
                         LD     HL,4000H        ;HL = START OF SCREEN
                         LD     BC,6143         ;SIZE OF SCREEN ITEM
                         XOR    A               ;CLEAR SCREEN
                         LD     HL,             ;SET FIRST LINE FOR
                         LD     DE,
                         LD     (HL),7          ;   SEVEN
                         LD     HL,5800H        ;SET ATTRIBUTE
                         LD     DE,5801H        ;START FROM SECOND LINE
                         LD     BC,
```

```
        LD      (HL),A          ; PAPER 0100 + (INK 0

        LD      HL,5800H        ;SET HIGHWAY
        LD      DE,[illegible]  ;[illegible] SCT ON
        LD      BC,[illegible]
        LD      A,56            ;(PAPER 7100 + [illegible]
        [illegible]
        LD      B,32            ;FILL ONE LINE
        EXX
                [illegible]
        LD      (DE),A
        LD      (BC)
        [illegible] HL
        INC     DL
        [illegible]
        [illegible]
        [illegible]
        [illegible]
        RET

        PUSH    HL              ;SAVE HL PTR
        LD      A,[illegible]
        [illegible]
        [illegible]
        LD      D,0
        LD      E,A
        ADD     HL,DE           ;PTR TO POS OF SHAPE
        LD      E,(HL)          ;DE RETURN SHAPE PTR
        INC     HL
        LD      D,(HL)
        POP     HL

;******** [illegible] ********

        display ASCII value from character set
        [illegible] - store DE, the message pointer
                [illegible]    the same after display
                               register as well

        PUSH    HL
        PUSH    DE
        [illegible] HL
        LD      A,(DE)          ;LOAD ASCII CHAR
        [illegible]
        ADD     HL,HL           ;MULTIPLE OF 8 BYTES
        [illegible]
        ADD     HL,HL
        [illegible]
        LD      HL,CHARSET      ;START OF CHARACTER SET
        [illegible]
        [illegible]
```

```
        AND     [illegible]
        JR      [illegible]
        POP     [illegible]
        CALL    [illegible]
        AND     [illegible]
        CP      [illegible]
        RET     [illegible]
        LD      [illegible]
        CALL    [illegible]
        AND
        JR      [illegible]
```

```
[illegible]
        SPELF   [illegible]
[illegible]
                                        ;RESTORE CHAR
[illegible]
                                        ;[illegible] ROW COUNT
                                        ;[illegible] ATTR
[illegible]
                LD      HL,[illegible]  ;ATTRIBUTE START [illegible]
[illegible]
        SPRATT  [illegible]
[illegible]
                ADD     HL,HL
[illegible]
                JR      Z,NXT6FA
```

```
09460 NX  [illegible]  POP     HL
[illegible]            LD      [illegible]
09480                  DEC     A
09490                  RET     Z
[illegible]            EX      [illegible]
095[illegible]         LD      E,32
[illegible]

09550 ;
09560 RESPC            LD      A,(PCAREST)         ;TEST PC E[illegible]
09570                  AND     [illegible]
09580                  RET     [illegible]
09590                  LD      [illegible]
09600                  LD      [illegible]
[illegible]
09620                  LDIR                        ;[illegible]
09630                  EX      DE,HL               ;DE IS [illegible] PTR
09640                  LD      HL,[illegible]      ;LOAD POS
09650                  LD      [illegible]
09660                  EX      [illegible]
09670 RPCLP1           PUSH    HL                  ;SAVE POS
09680                  LD      A,(SPTP)
09690                  LD      [illegible]
09700                  ADD     [illegible]
09710                  RET     [illegible]
09720                  JR      [illegible]
09730                  LD      [illegible]
09740                  ADD     [illegible]
09750                  LD      [illegible]
09760 N[illegible]     LD      [illegible]
[illegible]
09780                  JR      [illegible]
09790                  LD      [illegible]
09800 RP[illegible]    PUSH    HL
098[illegible]         LD      [illegible]
      RPCLP3           LD      A,(DE)              ;RESTORE CHAR
[illegible]

      NXTRPC           POP     [illegible]
[illegible]                                        ;[illegible] ROW COUNT
                                                   ;RESTORE POLICE CAR
[illegible]                                        ;[illegible] UP ONE LINE
```

```
                                                ;ATTR START LOADING POS

                                                ;RESET ... COUN

                        ; frog if any left
                        ; 0 if none left

LD      A,(FROGLFT)
AND     A
RET     N                                       ;RETURN IF ...
LD      HL,FROG...
```

```
        HL,FRGSTN       ;UPDATE FROG STATION
        HL              ;MOVE 3 CHARACTER LEFT

        HL

        HL,FRGSTR       ;INIT FRG STR FOR REG
        DE,FRGSTR+1     ;CLEAR FROG STORE

        A
        HL,0E020H       ;H=-32, L=32
        C,A             ;C = ARG MOVE
                        ;TEST RIGHT

                        ;TEST LEFT

                        ;ADD 3

                        ;TEST UP
```

```
                                        ;STORE TEMP

                                        ;FOR BOUNDARY ADJ

              AND
              CP
              JR
              LD
              AND
              SRL
              JR

                                        ;TEST WITHIN ROW

                                        ;TEST LAST FROG
```

```
75B8 08
75B9 A7
75BA ED52
75BD C8      11750            RET   Z             ;RETURN IF SAME
75BE CD0675  11760            CALL  RESFRG        ;RESTORE FROG
75C1 2A7A6F  11770            LD    HL,(NEWFRG)   ;UPDATE OLD FROG POS
75C4 227B6F
75C7 217D6F
75CA 117B6E
75CD 010500
75D0 EDB0
75D2 CD2876
     C9

75D6 118V60  11870 RESFRG     LD    DE,FRGSTR     ;STORAGE PTR
75D9 2A7B6F  11880            LD    HL,(OLDFRG)   ;RESTORE FROM OLDFRG
75DD 3E02    11900            LD    A,2           ;ROW COUNTER
75DF 08      11910            EX    AF,AF'
75E1 0E02    11930            LD    C,2           ;COLUMN COUNTER
75E8 13      11980            INC   DE
75E9 24      11990            INC   H             ;NEXT CHAR BYTE
75EE 0D      12030            DEC   C             ;COLUMN COUNT
75F3 3D      12070            DEC   A             ;ROW COUNT
                                                  ;UP ONE LINE
```

```
                                A               ;ROW COUNTER
0E07                                            ;RESTORE ATTR
EDB0
E8
E1
19
3D                                              ;UPDATE ROW COUNTER
C8
0E20
ED42
19EE
2A6F    12450         LD        HL,(NEWRNG)
09
```

```
[illegible]                                          ;LINE AD[illegible]
[illegible]   PUSH
[illegible]   PUSH
[illegible]   XOR                                    ;[illegible] AND [illegible]
[illegible]   LD
[illegible]   INC
[illegible]   LD
[illegible]   INC
[illegible]   LD
[illegible]   DEC
[illegible]   LD
[illegible]   CALL
[illegible]   POP
[illegible]   PUSH
[illegible]   EX
[illegible]   LD                                     ;[illegible] PAPER, [illegible]
[illegible]   INC
[illegible]   LD
[illegible]   AND
L D52 [illegible]
[illegible] 28 [illegible]  INC
[illegible]   LD
[illegible]   EX
[illegible]   CALL
[illegible] E1   POP
[illegible]   POP
[illegible] 10 DE   DJNZ
C9   RET

D9   EXX
E5   PUSH   HL
[illegible]   CALL
```

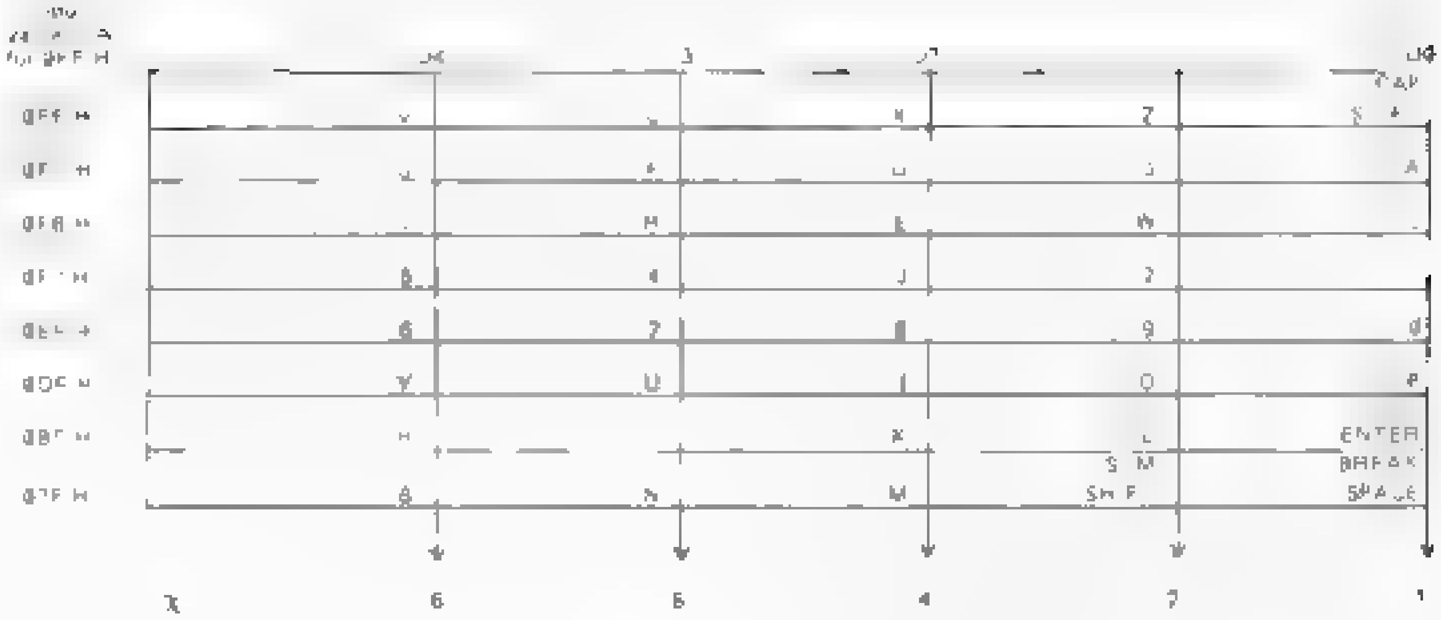
```
;MOVE UP DATABASE

;TEST FIRST BLOCK
;TEST LOW HIGHWAY
;NOT EVER STEP ON HWY

;TEST IN LOW HWY

;NO MORE IF STEP HWY
```

(18 POLICE CAR ON

```
77CC E5          15050 RANDMD   PUSH    HL
77CD C5          15060          PUSH    BC
77CE 2A756F      15070          LD      [illegible] RND
77D[illegible] 46          15080          LD      B,(HL)
[illegible]
                                        [illegible]        ;BOUND BY [illegible] WITHIN [illegible]
[illegible]
77D7 78          15130          LD      A,B
[illegible]
77DB C1          15150          POP     BC
77DC E1          15160          POP     HL
77DD C9          15170          RET
                 15180 ;
77DE 2A446F      15200 OVER     LD      HL,(SCORE+1)    ;HIGH SCORE MANAGE
77E1 [illegible]        [illegible]
77E4 [illegible]        15220          LD      [illegible]
77E6 1A          15230 [illegible]  LD      [illegible]
77E7 BE          15240          CP      (HL)
[illegible]
77EB [illegible]        15270          JP      [illegible]         ;UPDATE HIGH SCORE
77ED [illegible]        15280 [illegible]  INC     HL
77EF [illegible]        [illegible]          INC     [illegible]
[illegible]
77F2 2144[illegible]      [illegible]          LD      [illegible]
[illegible]
                                                        ;[illegible]
[illegible]
                                                        ;START OF [illegible]
[illegible]
                 15530 ;
[illegible]             15540          END     START
00000 Total errors
```

APPENDIX A

SPECTRUM KEY INPUT TABLE



NB: To trap a key
    1. Load A with INPUT VALUE of the corresponding row.
                                   Bottom Row

BREAK SPACE key

JR Z,keyset ;normal state is always

APPENDIX B

MEMORY IN HEX	ATTRIBUTE IN HEX	LINE		MEMORY IN HEX	ATTRIBUTE IN HEX
4000	5800	0		401F	581F
4020	5820	1		403F	583F
4040	5840	2		405F	585F
4060	5860	3		407F	587F
4080	5880	4		409F	589F
40A0	58A0	5		40BF	58BF
40C0	58C0	6		40DF	58DF
40E0	58E0	7		40FF	58FF
4800	5900	8		481F	591F
4820	5920	9		483F	593F
4840	5940	10		485F	595F
4860	5960	11		487F	597F
4880	5980	12		489F	599F
48A0	59A0	13		48BF	59BF
48C0	59C0	14		48DF	59DF
48E0	59E0	15		48FF	59FF
5000	5A00	16		501F	5A1F
5020	5A20	17		503F	5A3F
5040	5A40	18		505F	5A5F
5060	5A60	19		507F	5A7F
5080	5A80	20		509F	5A9F
50A0	5AA0	21		50BF	5ABF
50C0	5AC0	22		50DF	5ADF
50E0	5AE0	23		50FF	5AFF

APPENDIX C

SPECTRUM CHARACTER SET TABLE

HEX	HIGH	0	1	2	3	4	5	6	7
LOW	BITS	000	001	010	011	100	101	110	111
0	0000	NU	INK ctrl	SPACE	0	@	P		p
1	0001	NU	PAPER ctrl	!	1	A	Q	a	
2	0010	NU	FLASH ctrl	"	2	B	R		
3	0011	NU	BRIGHT ctrl	#	3	C	S	c	s
4	0100	NU	INVERSE ctrl	$	4	D	T	d	
5	0101	NU	OVER ctrl	%	5	E	U	e	u
6	0110	PRINT	AT ctrl	&	6	F	V		v
7	0111	EDIT	TAB ctrl	'	7	G	W	g	w
8	1000	cursor left	NU	(	8	H	X	h	x
9	1001	cursor right	NU	)	9	I	Y		y
A	1010	cursor down	NU	*		J	Z		z
B	1011	cursor up	NU	+	;	K	[	k	
C	1100	DELETE	NU	,	<	L	\		
D	1101	ENTER	NU	-	=	M	]	m	
E	1110	number	NU		>	N	↑	n	
F	1111	NU	NU	/	?	O	_		©

NON PRINTABLE (columns 0–1) — PRINTABLE (columns 2–7)

NB NU Not Used

APPENDIX D

DECIMAL HEXADECIMAL CONVERSION TABLES

HEX	0		2	3	4	5	6	7	8	9	A	B		D	E	F	X00	X000
0	[illegible]	[illegible]	2	3	4	5	6	[illegible]	8	9	0	[illegible]	2	13	14	[illegible]	0	0
	6	[illegible]	8	9	2	2	22	2	24	25	26	2	28	29	30	3	256	4096
	32	3	34	35	36	3	38	39	4	4	42	43	44	45	46	47	5 2	8 9
3	48	49	50	5	52	53	54	55	56	57	58	59	60	6	62	63	68	2288
4	64	65	66	6	68	69	70	[illegible]	7	73	4	75	6	[illegible]	78	79	024	6384
5	80	8	82	83	84	85	86	87	88	89	90	9	92	93	94	95	280	20480
6	96	97	98	99	00	0	02	03	104	05	06	0	108	09	10	11	536	24576
7	2	3	4	15	16	17	16	19	20	121	122	123	124	135	26	12	792	28672
8	128	129	30	3	32	133	34	35	36	137	38	139	40	4	142	143	2048	32768
9	144	45	46	47	148	149	150	5	52	53	154	155	56	5	58	159	2304	36864
A	60	6	62	6	164	65	66	67	68	69	70	7	172	173	74	175	2560	40960
B	76	77	78	79	80	8	182	83	84	85	186	187	188	189	90	9	2816	45056
	92	93	94	195	196	97	198	199	200	20	202	203	204	205	206	207	3072	49152
D	208	209	2 0	2	2 2	213	214	215	2 6	2	218	219	220	22	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	24	242	243	244	245	246	247	248	249	250	25	252	253	254	255	3840	61440

[illegible]

We have to determine the 16-bit binary number,

```
   ie  bbbbbbbb    bbbbbbbb
         HOB          LOB
```

[illegible]

four bits of the HOB (High Order Byte) to be 1 ie 01.

```
       0001bbbb    bbbbbbbb
         HOB          LOB
```

[illegible]

Since the difference is still greater than 255, we refer to the [illegible] under the column heading 00xx and find that 2104 is between 2048 and 2304. Again we take the lower value 2048 and arrive from the row value that the less significant four bytes of HOB is 8 ie 1000.

```
       00011000    bbbbbbbb
         HOB          LOB
```

[illegible] number. We find the difference between 2104 and 2048 as 56. From the large middle big sub-table we find that 56 is at the intersection of row 3 and column 8. So we take the LOB as 38H.

```
       00011000    00111000
         HOB          LOB
```

So the HEX-value of the number 6200 is 1838H.

APPENDIX E

2's COMPLEMENT DECIMAL HEXADECIMAL CONVERSION TABLE

[illegible]			2		4		6		8	[illegible]	[illegible]	[illegible]				
8	[illegible]		[illegible]	[illegible]	4	[illegible]				[illegible]	[illegible]		[illegible]	[illegible]	[illegible]	
					26		[illegible]	25	24	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]
A	96	[illegible]	94	[illegible]	[illegible]	[illegible]	[illegible]	89	88	[illegible]	86	[illegible]	84	[illegible]	[illegible]	[illegible]
B	[illegible]	[illegible]	[illegible]		[illegible]	[illegible]	[illegible]		[illegible]			69	68	[illegible]	66	65
	64	63	62	[illegible]	[illegible]	[illegible]	58	[illegible]	[illegible]	[illegible]	54	[illegible]	[illegible]	[illegible]	[illegible]	49
	48	[illegible]	46	45	44	43	42	[illegible]	40	39	38		[illegible]	[illegible]	34	33
E	[illegible]	[illegible]	30	29	28	[illegible]	26	[illegible]	24	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
F	6	5	4				0	[illegible]	8		6	[illegible]	4	3	2	

APPENDIX F

HEXADECIMAL ADDITION TABLE

	0	[illegible]	2	3	4	5	6	[illegible]	8	9	A	B	C	D	E	F
0	0	[illegible]	2	3	4	5	6	[illegible]	8	9	A	B	C	D	E	F
[illegible]	[illegible]	[illegible]	[illegible]	4	5	6	[illegible]	8	9	A	B	C	D	[illegible]	F	10
2	2	[illegible]	4	5	6	7	8	9	A	B	[illegible]	D	E	F	10	[illegible]
3	3	4	5	6	7	8	9	A	B	[illegible]	D	[illegible]	F	10	11	12
4	4	5	6	[illegible]	8	9	A	B	C	[illegible]	E	F	10	11	12	13
5	5	6	[illegible]	8	9	A	B	C	D	E	F	10	[illegible]	12	13	14
6	6	[illegible]	8	9	A	B	[illegible]	D	E	[illegible]	10	11	12	13	14	15
[illegible]	[illegible]	8	9	A	B	[illegible]	[illegible]	E	F	10	[illegible]	12	[illegible]	14	15	16
8	8	9	A	B	[illegible]	[illegible]	E	F	[illegible]	[illegible]	12	13	14	15	16	[illegible]
9	9	A	B	[illegible]	D	E	F	10	[illegible]	12	13	14	15	16	17	18
A	A	B	[illegible]	D	E	F	10	[illegible]	12	13	14	15	16	17	18	19
B	B	[illegible]	D	E	F	10	[illegible]	12	13	14	15	16	[illegible]	18	19	1A
[illegible]	[illegible]	D	E	F	10	[illegible]	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	[illegible]	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	[illegible]	12	13	14	15	16	[illegible]	18	19	1A	1B	[illegible]	1D	1E

APPENDIX G

FLAG OPERATION SUMMARY TABLE

INSTRUCTION	C	Z	P/V	S	N	H	COMMENTS
ADC HL, SS	#	#	V	#	0	X	16-bit add with carry
ADC s; ADD s	#	#	V	#	0	#	8-bit add or add with carry
ADD DD, SS	#		–	–	0	X	16-bit add
AND s	0	#	P	#	0	1	Logical operations
BIT b, s	–	#	X	X	0	1	State of bit b of location s is copied into the Z flag
CCF	#				0	X	Complement carry
CPD; CPDR; CPI; CPIR		#	#	X	1	X	Block search instruction Z=1 if A=(HL), else Z=0 P/V=1 if BC≠0, otherwise P/V=0
CP s	#	#	V	#	1	#	Compare accumulator
CPL					1	1	Complement accumulator
DAA	#	#	P	#		#	Decimal adjust accumulator
DEC s	–	#	V	#	1	#	8-bit decrement
IN r, (C)	–	#	P	#	0	0	Input register indirect
INC s	–	#	V	#	0	#	8-bit increment
IND; INI	–	#	X	X	1	X	Block input Z=0 if B≠0 else Z=1
INDR; INIR	–	1	X	X	1	X	Block input Z=0 if B≠0 else Z=1
LD A, I; LD A, R		#	FF	#	0	0	Content of interrupt enable Flip Flop is copied into the P/V flag
LDD; LDI	–	X	#	X	0	0	Block transfer instructions
LDDR; LDIR		X	0	X	0	0	P/V=1 if BC≠0, otherwise P/V=0
NEG	#	#	V	#	1	#	Negate accumulator
OR s	0	#	P	#	0	0	Logical OR accumulator
OTDR; OTIR	–	1	X	X	1	X	Block output, Z=0 if B≠0 otherwise Z=1
OUTD; OUTI		#	X	X	1	X	Block output Z=0 if B≠0 otherwise Z=1
RLA; RLCA; RRA; RRCA	#				0	0	Rotate accumulator
RLD; RRD		#	P	#	0		Rotate digit left and right
RL s; RLC s; RR s; RRC s; SLA s; SRA s; SRL s	#	#	P	#	0	0	Rotate and shift location s
SBC HL, SS	#	#	V	#	1	X	16-bit subtract with carry
SCF	1				0	0	Set carry
SBC s; SUB s			V		1		8-bit subtract with carry
XOR s	0		P		0	0	Exclusive OR accumulator

	OPERATION
	Carry flag. C=1 if the operation produced a carry from the most significant bit of the operand or result.
	Zero flag. Z=1 if the result of the operation is
	Sign flag S 1 if the most significant bit of the result is one, ie a negative number
P/V	Parity or overflow flag. Parity (P) and overflow (O) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P V 1 if the result of the operation is even, P V 0 if result is odd. If P/V holds overflow, P V 1 if the result of the operation produced an overflow.
	H carry flag. H 1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator
	Add Subtract flag. N 1 if the previous operations was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtractionusing operands with packed BCD format.
	The flag is affected according to the result of the operation.
	The flag is unchanged by the operation
	The flag is reset (0) by the operation
	The flag is set (=1) by the operation.
	The flag result is unknown.
	The P V flag is affected according to the overflow result of the operation.
	P/V flag is affected according to the parity result of the operation.
	Any one of the CPU registers A,B,C,D,E,H,L.
	Any 8-bit location for all the addressing modes allowed for the particular instructions.
	Any 16-bit location for all the addressing modes allowed for that instruction.
	Refresh register
	8-bit value in range 0-255.
	16-bit value in range 0-65535

Z80 CPU INSTRUCTIONS SORTED BY OP-CODE

HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC
00	NOP	49	LD C,C	92	SUB D
01XXXX	LD BC,NN	4A	LD C,D	93	SUB E
02	LD (BC),A	4B	LD C,E	94	SUB H
03	INC BC	4C	LD C,H	95	SUB L
04	INC B	4D	LD C,L	96	SUB (HL)
05	DEC B	4E	LD C,(HL)	97	SUB A
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
07	RLCA	50	LD D,B	99	SBC A,C
08	EX AF,AF'	51	LD D,C	9A	SBC A,D
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	54	[illegible]	[illegible]	[illegible]
0C	INC C	55	LD D,L	9E	SBC A,(HL)
0D	DEC C	56	LD D,(HL)	9F	SBC A,A
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	AND B
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	AND C
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	AND H
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	AND A
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	XOR B
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	XOR C
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	XOR D
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	XOR E
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	XOR H
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	XOR A
[illegible]	[illegible]	[illegible]	LD H,A	B0	OR B
[illegible]	[illegible]	[illegible]	LD L,B	B1	OR C
[illegible]	[illegible]	[illegible]	LD L,C	B2	OR D
[illegible]	[illegible]	[illegible]	LD L,D	B3	OR E
[illegible]	[illegible]	[illegible]	LD L,E	B4	OR H
[illegible]	[illegible]	[illegible]	LD L,H	B5	OR L
[illegible]	[illegible]	[illegible]	LD L,L	B6	OR (HL)
[illegible]	[illegible]	[illegible]	LD L,(HL)	B7	OR A
[illegible]	[illegible]	[illegible]	LD L,A	B8	CP B
[illegible]	[illegible]	[illegible]	LD (HL),B	B9	CP C
[illegible]	[illegible]	[illegible]	LD (HL),C	BA	CP D
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	CP E
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	CP H
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2C	INC L	75	LD (HL),L	BE	CP (HL)
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2F	CPL	78	LD A,B	C1	POP BC
30XX	JR NC,DIS	79	LD A,C	C2XXXX	JP NZ,NN
31XXXX	LD SP,NN	7A	LD A,D	C3XXXX	JP NN
32XXXX	LD (NN),A	7B	LD A,E	C4XXXX	CALL NZ,NN
33	INC SP	7C	LD A,H	C5	PUSH BC
34	INC (HL)	7D	LD A,L	C6XX	ADD A,N
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
36XX	LD (HL),N	7F	LD A,A	C8	RET Z
37	SCF	80	ADD A,B	C9	RET
38XX	JR C,DIS	81	ADD A,C	CAXXXX	JP Z,NN
[illegible]	[illegible]	82	ADD A,D	CCXXXX	CALL Z,NN
3AXXXX	LD A,(NN)	83	ADD A,E	CDXXXX	CALL NN
3B	DEC SP	84	ADD A,H	CEXX	ADC A,N
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3EXX	LD A,N	87	ADD A,A	D1	POP DE
3F	CCF	88	ADC A,B	D2XXXX	JP NC,NN
40	LD B,B	89	ADC A,C	D3XX	OUT (N),A
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
43	LD B,E	8C	ADC A,H	D6XX	SUB N
44	LD B,H	8D	ADC A,L	D7	RST 10H
45	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
46	LD B,(HL)	8F	ADC A,A	D9	EXX
47	LD B,A	90	[illegible]	[illegible]	[illegible]
48	LD C,B	91	SUB C	DBXX	IN A,(N)

HEXADECIMAL	MNEMONIC
[illegible] XXXX	[illegible]
[illegible] XX	SBC A,N
[illegible]	RST 18H
[illegible]	[illegible]
[illegible]	POP HL
[illegible] XXXX	[illegible] NN
[illegible]	EX (SP),HL
[illegible] XXXX	CALL PO,NN
[illegible]	[illegible]
[illegible] XX	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible] XXXX	[illegible]
[illegible]	[illegible]
[illegible] XXXX	[illegible] NN
[illegible] XX	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible] XXXX	[illegible] NN
[illegible] XXXX	[illegible] NN
[illegible]	[illegible] A
[illegible] XX	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible] XXXX	[illegible]
[illegible]	[illegible]
[illegible] XXXX	[illegible] NN
[illegible] XX	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
... [illegible rows] ...	[illegible]
[illegible]	S A B
[illegible]	S A [illegible]
[illegible]	S A D
[illegible]	S A E
[illegible]	S A H
[illegible]	S A [illegible]
[illegible]	S A H
[illegible]	[illegible] A A

HEXADECIMAL	MNEMONIC
[illegible]	SRA B
CB29	SRA C
CB2A	SRA D
CB2B	SRA E
CB2C	SRA H
CB2D	SRA L
CB2E	SRA (HL)
CB2F	SRA A
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible] A
[illegible]	[illegible] 0,B
[illegible]	[illegible] 0,C
[illegible]	[illegible] 0,D
[illegible]	[illegible] 0,E
[illegible] 44	[illegible] 0,H
[illegible] 45	[illegible] 0,L
[illegible]	[illegible] 0,(HL)
[illegible]	[illegible] 0,A
... [illegible rows] ...	[illegible]
[illegible]	[illegible] 2,D
[illegible]	[illegible] 2,E
[illegible]	[illegible] 2,H
[illegible]	[illegible] 2,L
[illegible]	[illegible] 2,(HL)
[illegible]	[illegible] 2,A
[illegible]	[illegible] 3,B
[illegible]	[illegible] 3,C
[illegible]	[illegible] 3,D
[illegible]	[illegible] 3,E
[illegible]	[illegible] 3,H
[illegible]	[illegible] 3,L
[illegible]	[illegible] 3,(HL)
[illegible]	[illegible] 3,A
[illegible]	[illegible] 4,B
[illegible]	[illegible] 4,C
[illegible]	[illegible] 4,[illegible]
[illegible]	[illegible] 4,[illegible]
[illegible]	[illegible] 4,[illegible]
[illegible]	[illegible] 4,L
[illegible]	[illegible] 4,(HL)
[illegible]	[illegible] 4,A
[illegible]	[illegible] B
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	BIT 5,E
[illegible]	BIT 5,H
[illegible]	BIT 5,[illegible]
[illegible]	[illegible] HL
[illegible]	[illegible] A
[illegible]	[illegible] 6,B
[illegible]	[illegible] 6,C
BIT7	[illegible] 6,D
[illegible]	[illegible] 6,E
[illegible]	[illegible] 6,H
[illegible]	BIT 6,L
[illegible] 6	BIT 6 (HL)
[illegible]	BIT 6 A
CB78	BIT 7 B

HEXADECIMAL	MNEMONIC
[illegible]	[illegible]
[illegible]	[illegible]
... [illegible rows] ...	[illegible]
CB9C	[illegible]
CB9D	[illegible]
CB9E	[illegible]
CB9F	[illegible] A
CBA0	[illegible] 4 H
CBA1	[illegible] 4
CBA2	[illegible] 4
CBA3	[illegible] E
CBA4	[illegible] H
CBA5	[illegible] 4
CBA6	[illegible] 4
CBA7	[illegible] 4 [illegible]
[illegible]	[illegible]
... [illegible rows] ...	[illegible]
CBB7	[illegible] A
CBB8	[illegible] B
CBB9	[illegible]
CBBA	[illegible]
CBBB	[illegible]
CBBC	[illegible]
CBBD	[illegible]
CBBE	[illegible] HL
CBBF	[illegible] A
CBC0	SET [illegible] B
CBC[illegible]	S[illegible] C
CBC2	SET [illegible]

HEXADECIMAL	MNEMONIC
CBC6	[illegible]
CBC7	[illegible]
CBC8	[illegible]
CBC9	[illegible]
CBCA	[illegible]
CBCB	[illegible]
[illegible]	SET 2,L
[illegible]	SET 2,(HL)
[illegible]	SET 3,B
[illegible]	SET 3,D
[illegible]	SET 3,E
[illegible]	SET 3,H
[illegible]	LD (IX+d),N
[illegible]	ADD IX,SP
46XX	[illegible]

HEXADECIMAL	MNEMONIC
[illegible]	SBC A,(IX+d)
[illegible]	AND (IX+d)
DDAEXX	XOR (IX+d)
DDB6XX	OR (IX+d)
DDE1	POP IX
DDE5	PUSH IX
DDE9	JP (IX)
DDF9	LD SP,IX
ED51	OUT (C),D

HEXADECIMAL	MNEMONIC
ED6A	[illegible]
ED6BXXXX	[illegible]
ED6F	[illegible]
ED72	[illegible]
ED78	[illegible]
ED7A	[illegible]
ED7BXXXX	[illegible]
FDAEXX	XOR (IY+d)

HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
	PUSH [illegible]				
[illegible]					
[illegible]	LD SP,IY				
[illegible]	RLC (IY+d)				
[illegible]	[illegible]				
FDCBXX18	[illegible]				
FDCBXX1E	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	BIT 2,(IY+d)				
	BIT 3,(IY+d)				
[illegible]	BIT [illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]					
[illegible]	RES 0,(IY+d)				
[illegible]					
[illegible]					
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]					
[illegible]					
[illegible]					
[illegible]					
[illegible]					
[illegible]	[illegible]				
[illegible]					
	[illegible]				
[illegible]	[illegible]				

APPENDIX I

Z80-CPU INSTRUCTIONS SORTED BY MNEMONIC

MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL
ADC A,(HL)	8E	BIT 2,B	CB 50	[illegible]	FE XX
ADC A,(IX+dis)	DD 8E XX	BIT 2,C	[illegible]	CP E	BB
ADC A,(IY+dis)	FD 8E XX	BIT 2,D	CB 52	CP H	[illegible]
[illegible]	8F	BIT 2,E	CB 53	CP L	[illegible]
[illegible]	88	BIT 2,H	CB 54	[illegible]	ED A9
[illegible]	89	[illegible]	CB 55	[illegible]	ED B9
[illegible]	8A	BIT 3,(HL)	CB 5E	[illegible]	ED A1
[illegible]	CE XX	BIT 3,(IX+dis)	DD CB XX 5E	[illegible]	ED B1
[illegible]	8B	BIT 3,(IY+dis)	FD CB XX 5E	[illegible]	2F
[illegible]	8C	BIT 3,A	CB 5F	DAA	27
[illegible]	8D	BIT 3,B	CB 58	[illegible]	35
[illegible]	ED 4A	[illegible]	CB 59	[illegible]	DD 35 XX
[illegible]	ED 5A	[illegible]	CB 5A	[illegible]	FD 35 XX
[illegible]	ED 6A	[illegible]	CB 5B	[illegible]	[illegible]
[illegible]	ED 7A	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	86	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	DD 86 XX	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	FD 86 XX	BIT 4,(IX+dis)	DD CB XX 66	[illegible]	[illegible]
[illegible]	87	BIT 4,(IY+dis)	FD CB XX 66	[illegible]	[illegible]
[illegible]	80	BIT 4,A	CB 67	[illegible]	[illegible]
[illegible]	81	BIT 4,B	CB 60	[illegible]	[illegible]
[illegible]	82	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	C6 XX	[illegible]	[illegible]	[illegible]	DD 2B
[illegible]	83	[illegible]	[illegible]	[illegible]	FD 2B
[illegible]	84	[illegible]	CB 64	[illegible]	[illegible]
[illegible]	85	[illegible]	CB 65	DEC SP	[illegible]
[illegible]	09	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	19	BIT 5,(IX+dis)	DD CB XX 6E	DJNZ,dis	10 XX
[illegible]	29	BIT 5,(IY+dis)	FD CB XX 6E	[illegible]	[illegible]
[illegible]	39	BIT 5,A	CB 6F	EX (SP),HL	E3
[illegible]	DD 09	BIT 5,B	CB 68	EX (SP),IX	DD E3
[illegible]	DD 19	BIT 5,C	CB 69	EX (SP),IY	FD E3
[illegible]	DD 29	BIT 5,D	CB 6A	EX AF,AF'	08
[illegible]	DD 39	BIT 5,E	CB 6B	EX DE,HL	EB
[illegible]	[illegible]	[illegible]	[illegible]	EXX	D9
[illegible]	FD 19	BIT 5,L	CB 6D	HALT	76
[illegible]	FD 29	BIT 6,(HL)	CB 76	IM 0	ED 46
[illegible]	FD 39	BIT 6,(IX+dis)	DD CB XX 76	IM 1	ED 56
[illegible]	A6	BIT 6,(IY+dis)	FD CB XX 76	IM 2	[illegible]
[illegible]	DD A6 XX	BIT 6,A	CB 77	IN A,(C)	[illegible]
[illegible]	FD A6 XX	BIT 6,B	CB 70	IN A,(N)	DB XX
[illegible]	A7	BIT 6,C	CB 71	IN B,(C)	ED 40
[illegible]	A0	BIT 6,D	CB 72	IN C,(C)	ED 48
[illegible]	A1	BIT 6,E	CB 73	IN D,(C)	ED 50
[illegible]	A2	BIT 6,H	CB 74	IN E,(C)	ED 58
[illegible]	E6 XX	BIT 6,L	CB 75	IN H,(C)	ED 60
[illegible]	A3	BIT 7,(HL)	CB 7E	IN L,(C)	ED 68
[illegible]	A4	BIT 7,(IX+dis)	DD CB XX 7E	INC (HL)	34
[illegible]	A5	BIT 7,(IY+dis)	FD CB XX 7E	INC (IX+dis)	DD 34 XX
[illegible]	CB 46	BIT 7,A	CB 7F	INC (IY+dis)	FD 34 XX
[illegible]	DD CB XX 46	BIT 7,B	CB 78	INC A	3C
[illegible]	FD CB XX 46	BIT 7,C	CB 79	INC B	04
[illegible]	CB 47	BIT 7,D	CB 7A	INC BC	03
[illegible]	CB 40	BIT 7,E	CB 7B	INC C	0C
[illegible]	CB 41	BIT 7,H	CB 7C	INC D	14
[illegible]	CB 42	BIT 7,L	CB 7D	INC DE	13
[illegible]	CB 43	CALL ADDR	CD XX XX	INC E	[illegible]
[illegible]	CB 44	CALL C,ADDR	DC XX XX	INC H	24
[illegible]	CB 45	CALL M,ADDR	FC XX XX	INC HL	23
[illegible]	CB 4E	CALL NC,ADDR	D4 XX XX	INC IX	DD 23
[illegible]	DD CB XX 4E	CALL NZ,ADDR	C4 XX XX	INC IY	FD 23
[illegible]	FD CB XX 4E	CALL P,ADDR	F4 XX XX	INC L	2C
[illegible]	CB 4F	CALL PE,ADDR	EC XX XX	INC SP	33
[illegible]	CB 48	CALL PO,ADDR	E4 XX XX	IND	ED AA
[illegible]	CB 49	CALL Z,ADDR	CC XX XX	INDR	ED BA
[illegible]	CB 4A	CCF	3F	[illegible]	ED A2
[illegible]	CB 4B	CP (HL)	BE	INIR	ED B2
[illegible]	CB 4C	CP (IX+dis)	DD BE XX	JP (HL)	E9
[illegible]	CB 4D	CP (IY+dis)	FD BE XX	JP (IX)	DD E9
[illegible]	CB 56	CP A	BF	JP (IY)	FD E9
[illegible]	DD CB XX 56	CP B	B8	JP ADDR	C3 XX XX
[illegible]	FD CB XX 56	CP C	B9	JP C,ADDR	DA XX XX
[illegible]	CB 57	CP D	BA	JP M,ADDR	FA XX XX

MNEMONIC	HEXADECIMAL
LD (ADDR),HL	ED 63 XX XX
LD (ADDR),HL	22 XX XX
LD (ADDR),IX	DD [illegible] XX XX
LD (ADDR),SP	ED 73 XX XX
LD (BC),A	02
LD (DE),A	12
LD (HL),A	77
[illegible]	70
LD (HL),C	71
LD (HL),D	72
[illegible]	36 XX
LD (IX+dis),A	DD 77 XX
LD (IX+dis),E	DD 73 XX
LD (IX+dis),H	DD 74 XX
LD (IX+dis),L	DD 75 XX
LD (IY+dis),A	FD 77 XX
LD (IY+dis),B	FD 70 XX
[illegible]	46
[illegible]	44
[illegible]	45
LD BC,(ADDR)	ED 4B XX XX

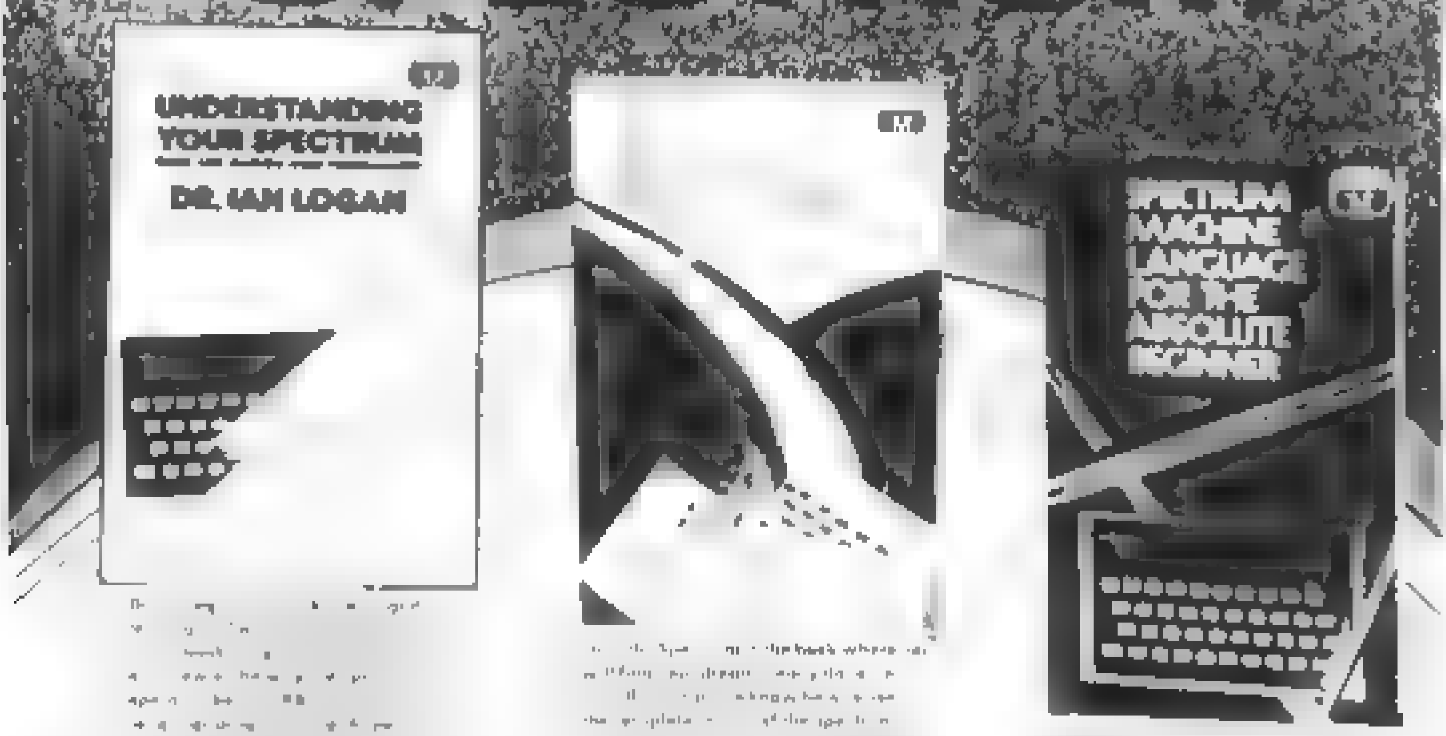
MNEMONIC	HEXADECIMAL
LD B,n	06 XX XX
LD C,(HL)	4E
[illegible]	DD 4E XX
[illegible]	FD 4E XX
[illegible]	4F
[illegible]	48
[illegible]	49
[illegible]	4A
[illegible]	0E XX
[illegible]	4B
LD D,(IY+dis)	FD 56 XX
LD D,A	57
LD D,D	52
[illegible]	16 XX
LD D,E	53
LD D,H	54
LD D,L	55
LD DE,nn	11 XX XX
LD E,(HL)	5E
LD E,B	58
LD E,H	5C
LD H,(HL)	66
LD H,(IX+dis)	DD 66 XX
LD H,(IY+dis)	FD 66 XX
LD H,C	61
LD H,E	63
LD H,H	64
LD H,L	65
LD HL,(ADDR)	ED 6B XX XX
LD HL,(ADDR)	2A XX XX
LD HL,nn	21 XX XX
LD I,A	ED 47
LD IX,(ADDR)	DD 2A XX XX
LD IX,nn	DD 21 XX XX
[illegible]	6C
[illegible]	6D
[illegible]	ED 4F
LD SP,(ADDR)	ED 7B XX XX
LD SP,nn	31 XX XX
LD SP,HL	F9
LD SP,IX	DD F9
LD SP,IY	FD F9
[illegible]	ED A8

MNEMONIC	HEXADECIMAL
LDDR	ED B8
LDIR	ED B0
NEG	ED 44
NOP	00
[illegible]	DD B6 XX
[illegible]	FD B6 XX
OR H	B4
OR L	B5
OUT (C),A	ED 79
OUT (C),C	ED 49
OUT (C),D	ED 51
OUT (C),E	ED 59
OUT (C),L	ED 69
OUT port,A	D3 port
POP BC	C1
PUSH AF	F5
RES 0,(IX+dis)	[illegible]
RES 0,B	[illegible]
RES 0,C	[illegible]
RES 0,D	CB 82
RES 0,E	CB 83
RES 0,H	CB 84
RES 0,L	CB 85
RES 1,(HL)	CB 8E
RES 1,(IX+dis)	DD CB XX 8E
RES 1,(IY+dis)	FD CB XX 8E
[illegible]	CB 8C
[illegible]	CB 8D
[illegible]	CB 90
[illegible]	CB 91
[illegible]	CB 92
[illegible]	CB 93
RES 2,H	CB 94
RES 3,A	CB 9F

MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL
[illegible]	[illegible]	RLC C	CB 01	SET 1,L	CB CD
[illegible]	[illegible]	RLC D	CB 02	SET 2,(HL)	CB D6
[illegible]	[illegible]	RLC E	CB 03	SET 2,(IX+dis)	DD CB XX D6
[illegible]	[illegible]	RLC H	CB 04	SET 2,(IY+dis)	FD CB XX D6
[illegible]	[illegible]	RLC L	CB 05	SET 2,A	CB D7
[illegible]	[illegible]	RLCA	07	SET 2,B	CB D0
[illegible]	CB A6	RLD	ED 6F	SET 2,C	CB D1
[illegible]	DD CB XX A6	RR (HL)	CB 1E	SET 2,D	CB D2
[illegible]	[illegible] XX A6	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	CB A7	RR (IY+dis)	FD CB XX 1E	SET 2,H	CB D4
[illegible]	CB A0	RR A	CB 1F	SET 2,L	CB D5
[illegible]	CB A1	RR B	CB 18	SET 3,(HL)	CB DE
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	CB A3	RR D	CB 1A	SET 3,(IY+dis)	FD CB XX DE
[illegible]	CB A4	RR E	CB 1B	SET 3,A	CB DF
[illegible]	CB A5	RR H	CB 1C	SET 3,B	CB D8
[illegible]	CB AE	RR L	CB 1D	SET 3,C	CB D9
[illegible]	DD CB XX AE	RRA	1F	SET 3,D	CB DA
[illegible]	FD CB XX AE	RRC (HL)	CB 0E	SET 3,E	CB DB
[illegible]	CB AF	RRC (IX+dis)	DD CB XX 0E	SET 3,H	CB DC
[illegible]	CB A8	RRC (IY+dis)	FD CB XX 0E	SET 3,L	CB DD
[illegible]	CB A9	RRC A	CB 0F	SET 4,(HL)	CB E6
[illegible]	CB AA	RRC B	CB 08	SET 4,(IX+dis)	DD CB XX E6
[illegible]	CB AB	RRC C	CB 09	SET 4,(IY+dis)	FD CB XX E6
[illegible]	CB AC	RRC D	CB 0A	SET 4,A	CB E7
[illegible]	CB AD	RRC E	CB 0B	SET 4,B	CB E0
[illegible]	CB B6	RRC H	CB 0C	SET 4,C	CB E1
[illegible]	[illegible] XX B6	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	FD CB XX B6	RRCA	0F	SET 4,E	CB E3
[illegible]	CB B7	RRD	ED 67	SET 4,H	CB E4
[illegible]	CB B0	RST 00	C7	SET 4,L	CB [illegible]
[illegible]	CB B1	RST 08	CF	SET 5,(HL)	CB EE
[illegible]	CB B2	RST 10	D7	SET 5,(IX+dis)	DD CB XX EE
[illegible]	CB B3	RST 18	DF	SET 5,(IY+dis)	FD CB XX EE
[illegible]	CB B4	RST 20	E7	SET 5,A	CB EF
[illegible]	CB B5	RST 28	EF	SET 5,B	CB E8
[illegible]	CB BE	RST 30	F7	SET 5,C	CB E9
[illegible]	DD CB XX BE	RST 38	FF	SET 5,D	CB EA
[illegible]	FD CB XX BE	SBC A,(HL)	9E	SET 5,E	CB EB
[illegible]	CB BF	SBC A,(IX+dis)	DD 9E XX	SET 5,H	CB EC
[illegible]	CB B8	SBC A,(IY+dis)	FD 9E XX	SET 5,L	CB ED
[illegible]	CB B9	SBC A,A	9F	SET 6,(HL)	CB F6
[illegible]	CB BA	SBC A,B	98	SET 6,(IX+dis)	DD CB XX F6
[illegible]	CB BB	SBC A,C	99	SET 6,(IY+dis)	FD CB XX F6
[illegible]	CB BC	SBC A,D	9A	SET 6,A	CB F7
[illegible]	CB BD	SBC A,n	DE XX	SET 6,B	CB F0
[illegible]	C9	SBC A,E	9B	SET 6,C	CB F1
[illegible]	D8	SBC A,H	9C	SET 6,D	CB F2
[illegible]	F8	SBC A,L	9D	SET 6,E	CB F3
[illegible]	D0	SBC HL,BC	ED 42	SET 6,H	CB F4
[illegible]	C0	SBC HL,DE	ED 52	SET 6,L	CB F5
[illegible]	F0	SBC HL,HL	ED 62	SET 7,(HL)	CB FE
[illegible]	E8	SBC HL,SP	ED 72	SET 7,(IX+dis)	DD CB XX FE
[illegible]	E0	SCF	37	SET 7,(IY+dis)	FD CB XX FE
[illegible]	C8	SET 0,(HL)	CB C6	SET 7,A	CB FF
[illegible]	ED 4D	SET 0,(IX+dis)	DD CB XX C6	SET 7,B	CB F8
[illegible]	ED 45	SET 0,(IY+dis)	FD CB XX C6	SET 7,C	CB F9
[illegible]	CB 16	SET 0,A	CB C7	SET 7,D	CB FA
[illegible]	DD CB XX 16	SET 0,B	CB C0	SET 7,E	CB FB
[illegible]	FD CB XX 16	SET 0,C	CB C1	SET 7,H	CB FC
[illegible]	CB 17	SET 0,D	CB C2	SET 7,L	CB FD
[illegible]	CB 10	SET 0,E	CB C3	SLA (HL)	CB 26
[illegible]	CB 11	SET 0,H	CB C4	SLA (IX+dis)	DD CB XX 26
[illegible]	CB 12	SET 0,L	CB C5	SLA (IY+dis)	FD CB XX 26
[illegible]	CB 13	SET 1,(HL)	CB CE	SLA A	CB 27
[illegible]	CB 14	SET 1,(IX+dis)	DD CB XX CE	SLA B	CB 20
[illegible]	CB 15	SET 1,(IY+dis)	FD CB XX CE	SLA C	CB 21
[illegible]	17	SET 1,A	CB CF	SLA D	CB 22
[illegible]	CB 06	SET 1,B	CB C8	SLA E	CB 23
RLC (IX+dis)	DD CB XX 06	SET 1,C	CB C9	SLA H	CB 24
RLC (IY+dis)	FD CB XX 06	SET 1,D	CB CA	SLA L	CB 25
[illegible]	CB 07	SET 1,E	CB CB	SRA (HL)	CB 2E
[illegible]	CB 00	SET 1,H	CB CC	SRA (IX+dis)	DD CB XX 2E

MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL	MNEMONIC	HEXADECIMAL
SRA (IY+d)	[illegible] XX 2E				
SRA A	[illegible]				
SRA B	[illegible]				
SRA C	[illegible]				
SRA D	[illegible]				
SRA E	[illegible]				
SRA H	[illegible]				
SRA L	[illegible]				
SRL (HL)	[illegible]				
[illegible]	[illegible] XX 3E				
SRL (IY+d)	[illegible] XX 3E				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	96				
[illegible]	[illegible] XX				
[illegible]	[illegible] XX				
[illegible]	[illegible]				
[illegible]	90				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible] XX				
[illegible]	94				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible] AE XX				
[illegible]	[illegible] AE XX				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible] XX				
[illegible]	[illegible]				
[illegible]	[illegible]				
[illegible]	[illegible]				

ANNOUNCING The BEST Books For Your SPECTRUM



After leading the way in Sinclair ZX81 software, we've produced the highest quality, most exciting Spectrum software available. From the three excellent books depicted above to fast action games on cassette we're providing the best choice in Sinclair Spectrum software today.

Whether it's for your new Spectrum or ZX81 Melbourne House has books and programs perfectly suited to your needs.

Send for your Spectrum or ZX81 catalogue today.

Cassette of all programs from Spectrum Machine Language book is available from Melbourne House.

MELBOURNE HOUSE PUBLISHERS

INDEX

A
Absolute address 38, 47
Absolute jump 101
Accumulator 34, 135
ADD 68, 69
Add with carry 96
adding 12, 15, 31, 126
Addresses 30, 45, 47, 54, 69, 83, 106, 35
Alphanumeric 24, 28
Alternate register 35, 36
AND 75, 76, 78, 136
[illegible]
Arithmetic logic unit 31
Arithmetic operations 62, 68, 95, 126
[illegible]
[illegible] 28
Assembly language 8, 9, 60
Attribute file 57, 65, 132, 142

B
BASIC 5-8, 10, 13, 23, 38, 130
BEEP 144
Binary 18, 24, 25
Binary coded decimal 62, 63, 126
Bit set 117
Bits 19, 21, 23, 24, 32, 34, 75
Block 108, 123, 134
Boolean operations 75, 77
BORCR 124
BUSRQ [illegible]
Byte 24, 29, 34, 45, 135

C
Calculations 11, 14, 15
Calculator 31
CALL 106
Carry 70, 119, 126
Carry arithmetic 97
Carry flag 60, 63, 70, 77, 98, 99, 105, 106
[illegible]

D
[illegible]

E
EBCDIC 25
Electrical signals
E[illegible] 124
E[illegible] 115
Exchange register 36, 115
E[illegible]
Execution 6, 10

F
Firmware [illegible]
Flags 32, 58, 66, 72, 96, 99
Flag register 34, 58
Freeway frog 161, 163
Frequency 125, 144

H
Half carry flag [illegible]
HALT
Hardware
Hexadecimal
HEXLOAD
High level language
High order byte
Highest bit
HL register [illegible]

I
Immediate addressing 44, 79
Immediate extended
[illegible]ing 55, 79, 81
Immediate indexed addressing 55
INC 8, 64
[illegible]
[illegible]
Indexed addressing 47
Indicator 56
Ink 57
Instruction register 31
Instruction set 57
Instructions 10, 58
INT 127
Integer 25, 29
Interrupt 14, 127
Interrupt vector register 37

J
JP 61, 63, 99
Jumps 99, 102

K
Keyboard 5, 122, 135

L
Labels
LD 43, 70, 86
Logical operators 75
Logical operations 62, 63
Loops 99
Loudspeaker 125, 144
Low byte 85, 89
Low order 136, 140

M
Matrix 135
Megahertz 30
Memory 6, 10, 14, 17, 29, 86, 87, 144
Mnemonics 8, 9, 13, 43
Modes 55, 79

N
Nanosecond 52
Negative numbers 25, 29
N M I 127
Numbers 8, 24

O
Operands 71
Operating system 5-7, 39, 41, 135
OR 75, 76, 78
Overflow 96

P
PAPER 57
Parity/overflow 62, 63, 66, 77, 97, 105, 106
PEEK 7
Pins 6, 30
Pointer 46, 47, 79
POP 14, 15, 17, 37, 81, 91, 98
Port 135, 144
Processor 30
Program counter 31, 100, 101, 127
PUSH 14, 15, 17, 37, 81, 91, 98

R
R register 37
RAM 31
Random number 37
Registers 12, 17, 32, 54, 69, 83
Register addressing 45, 48, 51, 80, 81
Register indirect addressing 46, 54, 80
Relative jump 101
Relocatable 47
RET 98, 106
RETI 127
RLA 119
RLCA 120
ROM 7, 10, 31, 39, 124, 128, 135
Rotate 119
RST 128

S
SBC 77
Shift 119
Sign flag 59, 63, 66, 77, 97, 105, 106
Signed integer 25, 29
Silicon chip 30
Sound 124, 125, 144
SUB 71
SUBC 71
Subroutines 106, 131
Subtracting 12, 15, 31, 126
Subtraction flag (negate) 62, 63, 66, 77
SRA 121
Stack 14, 15, 17, 91, 127
Stack pointer 36, 92, 96, 97
STKEND 97
Syntax 130

T
Top down 130
Translation 79
Two's complement 27, 29

U
ULA 124, 144
User register 31, 32, 38
USR 39, 41, 56, 88, 93, 98

V
Variable 8, 13, 14, 55
Video screen 138

W
WAIT 124

X
XOR 75, 76, 78

Z
Zero flag 58, 59, 63, 66, 77, 97, 99, 105, 106
Zero page 127
Z80A, Z80 5, 6, 30

SPECTRUM
MACHINE LANGUAGE FOR THE
ABSOLUTE BEGINNER

R E G I S T R A T I O N
C A R D

Please fill out this page and return it promptly in order that we may keep you informed of new software and special offers that arise. Simply cut along the dotted line and return it to the correct address selected from those overleaf.

Where did you learn of this product?

☐ Magazine. If so, which one?..........................

☐ Through a friend

☐ Saw it in a Retail Store

☐ Other. Please specify.................................

Which Magazines do you purchase?

Regularly:...

Occasionally:..

What Age are you?

☐ 10-15 ☐ 16-19 ☐ 20-24 ☐ Over 25

We are continually writing new material and would appreciate receiving your comments on our product.

How would you rate this book?

☐ Excellent	☐ Value for money
☐ Good	☐ Priced right
☐ Poor	☐ Overpriced

Please tell us what software you would like to see produced for your computer.

Name...

Address..

..................................Code.....................

PUT THIS IN A STAMPED ENVELOPE AND SEND TO:

In the United States of America return page to:
Melbourne House Software Inc., 347 Reedwood Drive,
Nashville TN 37217.

In the United Kingdom return page to:
Melbourne House (Publishers) Ltd., Melbourne House, Church Yard,
Tring, Hertfordshire, HP23 5LU

In Australia & New Zealand return page to:
Melbourne House (Australia) Pty. Ltd., Suite 4, 75 Palmerston Crescent,
South Melbourne, Victoria, 3205.

SPECTRUM

Melbourne
House

'Your best course is to work through a book such as William Tang's *Spectrum Machine Language For The Absolute Beginner*.'

'This book is one of the best I have seen on the subject — for once the title is right on the nose! I can recommend this to anyone just getting interested.' — Popular Computing Weekly.

If you are are frustrated by the limitations of BASIC and want to write faster, more powerful, space-saving programs or subroutines, then *Spectrum Machine Language For The Absolute Beginner* is the book for you.

Even with no previous experience of computer languages, you will be able to discover the ease and power of the Spectrum's own language. Each chapter includes specific examples of machine language applications which can be demonstrated and used on your Spectrum, as well as a self-test questionnaire.

At the end of the book, all this is brought together into an entire machine language program — from design right through to the complete listing of an exciting, original arcade game.

Melbourne
House
Publishers

ISBN 0-86161-110-1

9 780861 611102